

# Verification of Bit-Flip Attacks against Quantized Neural Networks

YEDI ZHANG, National University of Singapore, Singapore

LEI HUANG, ShanghaiTech University, China

PENGFEEI GAO, ByteDance Inc, China

FU SONG<sup>\*†‡</sup>, Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China

JUN SUN, Singapore Management University, Singapore

JIN SONG DONG, National University of Singapore, Singapore

In the rapidly evolving landscape of neural network security, the resilience of neural networks against bit-flip attacks (i.e., an attacker maliciously flips an extremely small amount of bits within its parameter storage memory system to induce harmful behavior), has emerged as a relevant area of research. Existing studies suggest that quantization may serve as a viable defense against such attacks. Recognizing the documented susceptibility of real-valued neural networks to such attacks and the comparative robustness of quantized neural networks (QNNs), in this work, we introduce BFAVerifier, the first verification framework designed to formally verify the absence of bit-flip attacks or to identify all vulnerable parameters in a sound and rigorous manner. BFAVerifier comprises two integral components: an abstraction-based method and an MILP-based method. Specifically, we first conduct a reachability analysis with respect to symbolic parameters that represent the potential bit-flip attacks, based on a novel abstract domain with a sound guarantee. If the reachability analysis fails to prove the resilience of such attacks, then we encode this verification problem into an equivalent MILP problem which can be solved by off-the-shelf solvers. Therefore, BFAVerifier is sound, complete, and reasonably efficient. We conduct extensive experiments, which demonstrate its effectiveness and efficiency across various network architectures, quantization bit-widths, and adversary capabilities.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Software and its engineering** → **Formal software verification**; • **Security and privacy** → **Software security engineering**.

Additional Key Words and Phrases: Bit-Flip Attacks, Quantized Neural Networks, Formal Verification, Robustness

## ACM Reference Format:

Yedi Zhang, Lei Huang, Pengfei Gao, Fu Song, Jun Sun, and Jin Song Dong. 2025. Verification of Bit-Flip Attacks against Quantized Neural Networks. 1, 1 (February 2025), 37 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

\*Corresponding author.

†Also with University of Chinese Academy of Sciences.

‡Also with Nanjing Institute of Software Technology.

Authors' Contact Information: Yedi Zhang, National University of Singapore, Singapore, Singapore, yd.zhang@nus.edu.sg; Lei Huang, ShanghaiTech University, Shanghai, China, huanglei@shanghaitech.edu.cn; Pengfei Gao, ByteDance Inc, Beijing, China, gaopengfei.se@bytedance.com; Fu Song, Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China, songfu@ios.ac.cn; Jun Sun, Singapore Management University, Singapore, Singapore, junsun@smu.edu.sg; Jin Song Dong, National University of Singapore, Singapore, Singapore, dcsdjs@nus.edu.sg.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM XXXX-XXXX/2025/2-ART

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

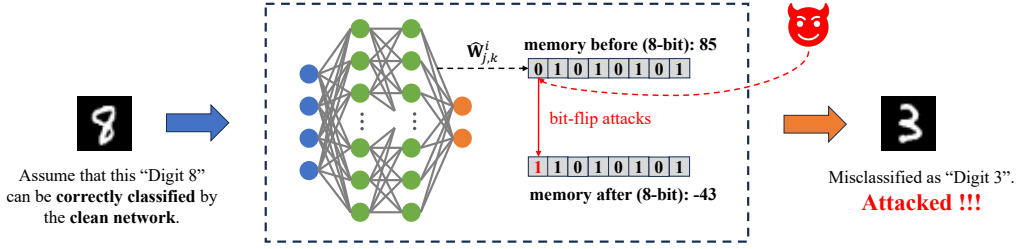


Fig. 1. An illustration example of bit-flip attacks on an 8-bit quantized neural network. The attacker flips a single bit in the final/output layer, altering the value of parameter  $\bar{W}_{j,k}^i$  from 85 to -43 (represented in two's complement) and misleading the network behavior.

## 1 Introduction

Neural networks have demonstrated their potential to achieve human-level performance in multiple domains [17, 53]. However, they are fragile in many ways and can be easily manipulated through various attacks [8, 10–12, 30, 38, 50, 67, 82, 89]. Recently, bit-flip attacks (BFAs) [16, 46, 59, 61] have become a critical class of hardware-based adversarial threats that exploit the physical vulnerability of neural networks. These attacks involve maliciously flipping the bits in the memory cells that store the parameters of a neural network during the deployment stage or changing the real-time activation values during the inference stage. Such attacks have been demonstrated to be feasible in practice for altering the behavior of networks in multiple cases [7, 70, 73]. For instance, RowHammer [54, 69] is one of the most widely used BFA methods which exploits a vulnerability in DRAM by repeatedly accessing memory rows to induce unintended bit flips in adjacent rows, compromising data integrity and security for network parameters. Unlike traditional software-level adversarial attacks, which typically require modifications to input data, BFAs directly target the underlying hardware (e.g., memory), making them particularly effective and difficult to defend against.

Modern DNNs, characterized by their large sizes and 32-bit floating-point parameters, face high computational and storage demands, hindering their deployment on resource-limited embedded devices. Quantization [24, 27, 33], reducing the precision of parameters and/or activation values, offers a promising solution to compress the network, and enables the deployment of quantized neural networks (QNNs) on such devices. For example, the Tesla-FSD chip [78] employs an 8-bit integer format to store all network weights. On the other hand, QNNs have been demonstrated to exhibit greater resilience to BFAs compared to their real-valued counterparts. Specifically, DNNs are highly susceptible to BFAs, with successful attack rates reaching nearly 99% [30], particularly through the manipulation of the exponential bit of compromised parameters. In response, numerous defense strategies have been proposed [39, 48, 68], leveraging parameter quantization to fortify network security against bit-flip attacks. Despite these measures, QNNs remain vulnerable to BFAs, as existing defense techniques fall short of providing formal security assurances against such attacks. This vulnerability underscores the critical need for developing a rigorous verification method to ascertain the absence of BFAs, ensuring the integrity and reliability of QNNs in security-sensitive applications. An illustration example of bit-flip attacks on an 8-bit QNN can be found in Figure 1.

**Main contributions.** In this work, we propose the first **Bit-Flip Attacks Verification** method (BFAVerifier) to efficiently and effectively verify if the bit-flip attacks are absent given a QNN, concerning a given input region, that is also sound and complete. It guarantees the safety of the QNN (such as robustness with respect to a specified input region) when facing potential bit-flip attacks. Given a QNN and an input region, BFAVerifier first conducts a novel reachability analysis

to compute an overapproximation of the output range of the network under the potential attacks. Such an analysis generates two outcomes: i) Proved, meaning the absence of the potential BFAs, or ii) Unknown, meaning that it fails to prove the absence of successful attacks possibly due to a conservative approximation of the abstraction throughout the reachability analysis process. If the result is Unknown, we further encode this bit-flip attacks verification problem into an equivalent mixed-integer linear programming (MILP) problem, which can be solved by off-the-shelf solvers.

A key technical challenge is how to conduct the reachability analysis for QNNs, given the interested input region and the threat of potential bit-flip attacks (i.e., some network parameters become symbolic with unknown values). To tackle the challenge, we propose SymPoly, an advanced abstract domain that is built on DeepPoly and is equipped with new abstract transformers specifically designed for handling symbolic parameters. Initially, symbolic parameters are determined with specific parameter intervals for the QNN concerning the potential bit-flip attacks. Subsequent reachability analysis can then be conducted on the modified QNN, which is equipped with symbolic parameters, using SymPoly. To enhance the precision of our reachability analysis results, we also propose two optimization strategies, namely, *sub-interval division* and *binary search strategy*, to reduce the precision loss that arises from the abstract transformation concerning large value discrepancies with a single interval.

**Experimental results.** We implement our method as an end-to-end tool that uses Gurobi [26] as the back-end MILP solver. We extensively evaluate it on a large set of verification tasks using multiple QNNs for the MNIST [41] and ACAS Xu [34] datasets, where the number of hidden neurons varies from 30 to 5120, the quantization bit-width of QNNs ranges from 4 to 8, and the number of bits for bit-flip attacks ranges from 1 to 4 bits. For the reachability analysis, we compare BFAVerifier with a naive method that iteratively generates a new QNN  $\mathcal{N}'$  for each possible bit-flip attack and verifies whether the  $\mathcal{N}'$  still preserves the robustness property within the given input region via DeepPoly. The experimental results show that our method is much more efficient than the naive method (up to 30x faster), successfully proving a similar number of verification tasks and even proving some tasks that return unknown by the naive method. Moreover, with the binary search strategy, we can prove even more tasks. The results also confirm the effectiveness of the MILP-based method, which can help verify many tasks that cannot be solved by SymPoly solely. The experimental results also show that BFAVerifier can verify the absence of BFAs for most of the benign neural networks in our benchmark.

Our contributions are summarized as follows.

- We propose a novel abstract domain SymPoly to conduct reachability analysis for neural networks with symbolic parameters soundly and efficiently;
- We introduced the first sound, complete, and reasonably efficient bit-flip attacks verification method BFAVerifier for QNNs by combining SymPoly and an MILP-based method;
- We implement BFAVerifier as an end-to-end tool and conduct an extensive evaluation of various verification tasks, demonstrating its effectiveness and efficiency.

**Outline.** Section 2 presents the preliminary. Section 3 defines our problem and a naive method for solving the problem based on DeepPoly is given. We present our method in Section 4. Section 5 reports experimental results. Section 6 discusses related work and finally, Section 7 concludes this work. Missing proofs can be found in the appendix.

## 2 Preliminary

We denote by  $\mathbb{R}$  (resp.  $\mathbb{N}$ ) the set of real (resp. integer) numbers. Given a positive integer  $n$ , we denote by  $[n]$  the set of positive integers  $\{1, 2, \dots, n\}$ . We use  $x, x', \dots$  to denote scalars,  $\mathbf{x}, \mathbf{x}', \dots$  to

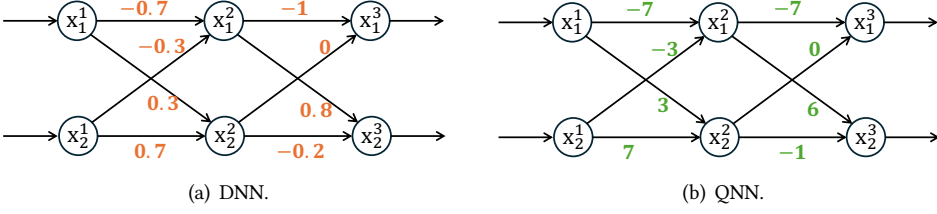


Fig. 2. A 3-layer DNN with ReLU activations and its quantized version.

denote vectors, and  $\mathbf{W}, \mathbf{W}', \dots$  to denote matrices. We denote by  $\mathbf{W}_{i,:}$  and  $\mathbf{W}_{:,j}$  to denote the  $i$ -th row and  $j$ -th column of the matrix  $\mathbf{W}$ , and use  $x_i$  to denote the  $i$ -th entry of the vector  $\mathbf{x}$ .

## 2.1 Neural Network and Quantization

In this section, we provide the minimal necessary background on neural networks and the quantization scheme considered in this work. Specifically, we focus on feedforward deep neural networks (DNNs) used for classification problems.

**Neural networks.** A DNN consists of an input layer, multiple hidden layers, and an output layer. Each layer contains neurons connected via weighted edges to the neurons in the subsequent layer. Specifically, each neuron in a non-input layer is additionally linked with a bias term. Given an input, a DNN computes an output by propagating it through the network layer by layer and gets the classification result by identifying the dimension with the highest value in the output vector.

A DNN with  $d$  layers can be represented by a non-linear multivariate function  $\mathcal{N} : \mathbb{R}^n \rightarrow \mathbb{R}^s$ . For any input  $\mathbf{x} \in \mathbb{R}^n$ , let  $\mathbf{x} = \mathbf{x}^1$ , the output  $\mathcal{N}(\mathbf{x}) = \mathbf{W}^d \mathbf{x}^{d-1} + \mathbf{b}^d$  can be obtained via the recursive definition  $\mathbf{x}^i = \text{ReLU}(\mathbf{W}^i \mathbf{x}^{i-1} + \mathbf{b}^i)$  for  $i \in \{2, 3, \dots, d-1\}$ , where  $\mathbf{W}^i$  and  $\mathbf{b}^i$  (for  $2 \leq i \leq d$ ) are the weight matrix and bias vector of the  $i$ -th layer. We refer to  $x_j^i$  as  $j$ -th neuron in the  $i$ -th layer and use  $n_i$  to denote the dimension of the  $i$ -th layer.  $n = n_1$  and  $s = n_d$ .

**Quantization.** Quantization is the process of converting high-precision floating-point values into a finite range of lower-precision ones, i.e., fixed-point numbers, without significant accuracy loss. A quantized neural network (QNN) is structurally similar to a DNN, except that the parameters and/or activation values are quantized into fixed-pointed numbers, e.g., 4-bit or 8-bit integers. In this work, we adopt the symmetric quantization scheme widely utilized in prior research concerning bit-blip attack (BFA) strategies on QNNs [16], where only parameters are quantized to reduce the memory requirements [27, 83, 91]. During inference, we assume that the parameters are de-quantized and all operations within the quantized networks are executed using floating-point arithmetic.

Given the weight matrix  $\mathbf{W}^i$  and the bias vector  $\mathbf{b}^i$ , their signed integer counterparts  $\widehat{\mathbf{W}}^i$  and  $\widehat{\mathbf{b}}^i$  with respect to quantization bit-width  $Q$  are respectively defined as follows. For each  $j, k$ ,

$$\widehat{\mathbf{W}}_{j,k}^i = \lfloor \mathbf{W}_{j,k}^i / \Delta w^i \rfloor, \quad \widehat{\mathbf{b}}_j^i = \lfloor \mathbf{b}_j^i / \Delta w^i \rfloor$$

where  $\Delta w^i = \text{maxAbs}(\mathbf{W}^i, \mathbf{b}^i) / (2^{Q-1} - 1)$  is the quantization step size of the  $i$ -th layer and the max function returns the maximal value of  $\mathbf{W}^i$  and  $\mathbf{b}^i$ .  $\lfloor \cdot \rfloor$  is the rounding operator,  $\text{maxAbs}(\mathbf{W}^i, \mathbf{b}^i)$  means finding the maximum absolute value among all the entries from  $\mathbf{W}^i$  and  $\mathbf{b}^i$ .

Once quantized into an integer, the parameter will be stored as the two's complement format in the memory. In the forward pass, the parameters will be de-quantized by multiplying the step size  $\Delta w^i$ . Taking a quantized parameter  $\widehat{\mathbf{W}}_{j,k}^i$  as an example and let  $\vec{v}(\cdot)$  denote the operation that converts an integer into its two's complement expressions. Assume that  $\vec{v}(\widehat{\mathbf{W}}_{j,k}^i) = [v_Q; v_{Q-1}; \dots; v_1]$ , then

the de-quantized version  $\widetilde{\mathbf{W}}_{j,k}^i$  can be calculated as follows with  $\widetilde{\mathbf{W}}_{j,k}^i \approx \mathbf{W}_{j,k}^i$ :

$$\widetilde{\mathbf{W}}_{j,k}^i = \widehat{\mathbf{W}}_{j,k}^i \cdot \Delta w^i = (-2^{Q-1} \cdot v_Q + \sum_{q=1}^{Q-2} 2^{q-1} \cdot v_q) \cdot \Delta w^i$$

*Example 2.1.* Consider the DNN shown in Figure 2(a). It contains three layers: one input layer, one hidden layer, and one output layer. The weights are associated with the edges and all the biases are 0 and the quantization bit-width  $Q = 4$ . Then, the step size of the parameter quantizer for each non-input layer is  $\Delta w^2 = 0.7/(2^3 - 1) = 0.1$ ,  $\Delta w^3 = 1/(2^3 - 1) = 1/7$ . The integer counterparts of weight parameters are associated with the edges in Figure 2(b).

Take the hidden layer for instance, we obtain their quantized weights, two's complement counterparts, and de-quantized versions as follows:

- $\widehat{\mathbf{W}}_{1,1}^2 = \lfloor -0.7/\Delta w^2 \rfloor = \lfloor -0.7 * 10 \rfloor = -7$ ,  $\vec{v}(\widehat{\mathbf{W}}_{1,1}^2) = [1001]$ , and  $\widetilde{\mathbf{W}}_{1,1}^2 = -0.7$ ;
- $\widehat{\mathbf{W}}_{1,2}^2 = \lfloor -0.3/\Delta w^2 \rfloor = \lfloor -0.3 * 10 \rfloor = -3$ ,  $\vec{v}(\widehat{\mathbf{W}}_{1,2}^2) = [1101]$ , and  $\widetilde{\mathbf{W}}_{1,2}^2 = -0.3$ ;
- $\widehat{\mathbf{W}}_{2,1}^2 = \lfloor 0.3/\Delta w^2 \rfloor = \lfloor 0.3 * 10 \rfloor = 3$ ,  $\vec{v}(\widehat{\mathbf{W}}_{2,1}^2) = [0011]$ , and  $\widetilde{\mathbf{W}}_{2,1}^2 = 0.3$ ;
- $\widehat{\mathbf{W}}_{2,2}^2 = \lfloor 0.5/\Delta w^2 \rfloor = \lfloor 0.7 * 10 \rfloor = 7$ ,  $\vec{v}(\widehat{\mathbf{W}}_{2,2}^2) = [0111]$ , and  $\widetilde{\mathbf{W}}_{2,2}^2 = 0.7$ ;

Similarly, for the output layer, we have

- $\widehat{\mathbf{W}}_{1,1}^3 = \lfloor -1/\Delta w^3 \rfloor = \lfloor -1 * 7 \rfloor = -7$ ,  $\vec{v}(\widehat{\mathbf{W}}_{1,1}^3) = [1001]$ , and  $\widetilde{\mathbf{W}}_{1,1}^3 = -1$ ;
- $\widehat{\mathbf{W}}_{1,2}^3 = \lfloor 0/\Delta w^3 \rfloor = \lfloor 0 * 7 \rfloor = 0$ ,  $\vec{v}(\widehat{\mathbf{W}}_{1,2}^3) = [0000]$ , and  $\widetilde{\mathbf{W}}_{1,2}^3 = 0$ ;
- $\widehat{\mathbf{W}}_{2,1}^3 = \lfloor 0.8/\Delta w^3 \rfloor = \lfloor 0.8 * 7 \rfloor = 6$ ,  $\vec{v}(\widehat{\mathbf{W}}_{2,1}^3) = [0110]$ , and  $\widetilde{\mathbf{W}}_{2,1}^3 = 0.8571$ ;
- $\widehat{\mathbf{W}}_{2,2}^3 = \lfloor -0.2/\Delta w^3 \rfloor = \lfloor -0.2 * 7 \rfloor = -1$ ,  $\vec{v}(\widehat{\mathbf{W}}_{2,2}^3) = [1111]$ , and  $\widetilde{\mathbf{W}}_{2,2}^3 = -0.1429$ .

## 2.2 Bit-Flip Attacks

Bit-flip attacks (BFAs) are a class of fault-injection attacks that were originally proposed to breach cryptographic primitives [3, 5, 6]. Recently, BFAs have been ported to neural networks.

**Attack scenarios and threat model.** Recent studies [40, 60, 82] have revealed vulnerabilities in DRAM chips, which act as a crucial memory component in hardware systems. Specifically, an adversary can induce bit-flips in memory by repeatedly accessing the adjacent memory rows in DRAM, without *direct* access to the victim model's memory, known as Rowhammer attack [40]. Such attacks exploit an unintended side effect in DRAM, where memory cells interact electrically by leaking charges, potentially altering the contents of nearby memory rows that were not originally targeted in the memory access. Although such attacks do not grant adversaries full control over the number or precise location of bit-flips and the most prevalent BFA tools such as DeepHammer [82] can typically induce only a single bit-flip, the recent study [16] has demonstrated that an adversary can effectively attack a QNN by flipping, on average, just one critical bit during the deployment stage. While indirectly flipping multiple bits is theoretically feasible, achieving this would require highly sophisticated techniques that are both extremely time-intensive and have a low likelihood of success in practice [60]. Therefore, in this study, we assume that the adversary can *indirectly* manipulate only a minimal number of parameters in a QNN, by default 1. More powerful attacks that can *directly* manipulate memory go beyond the scope of this work. On the other hand, though most of the existing BFAs target weights only [16, 30, 46, 61], in this work, we consider a more general setting where all parameters (weights and biases) of QNNs are vulnerable to BFAs [68].

*Example 2.2.* Consider the QNN given in Example 2.1. Suppose a bit-flip attacker can alter any single bit in the memory cell storing parameters and we use two dots “..” to represent a parameter that is targeted for such attacks. Take the parameter  $\widehat{\mathbf{W}}_{2,2}^3$  with  $\vec{v}(\widehat{\mathbf{W}}_{2,2}^3) = [1111]$  for example. Its

potential attacked representations are  $\vec{v}(\widehat{\mathbf{W}}_{2,2}^3) \in \{[0111], [1011], [1101], [1110]\}$ , thus we have  $\check{\mathbf{W}}_{2,2}^3 \in \{7, -5, -3, -2\}$  and  $\check{\mathbf{W}}_{2,2}^3 \in \{1, -0.7143, -0.4286, -0.2857\}$ . Given an input  $\mathbf{x}^1 = (1, 1)$ , after de-quantizing integer parameters during the inference, we can get the output of each non-input layer as  $\mathbf{x}^2 = (0, 1)$  and  $\mathbf{x}^3 = (0, -0.1429)$ .

Now, suppose that the attacker flips the fourth bit (i.e., sign bit) of the parameter  $\widehat{\mathbf{W}}_{2,2}^3$ , then we have  $\check{\mathbf{W}}_{2,2}^3 = 7$  and  $\check{\mathbf{W}}_{2,2}^3 = 1$ . Finally, the network output after the attack is  $\mathbf{x}^3 = (0, 1)$ , resulting in an altered classification outcome.

*Definition 2.3 (Attack Vector).* Given a QNN  $\mathcal{N}$  with quantization bit-width  $Q$ , and two integers  $m$  and  $n$  such that an adversary can attack any  $m$  parameters by flipping  $n$  bits at most within each parameter ( $n \leq Q$ ). An  $(m, n)$ -attack vector  $\rho$  is a set of pairs  $\{(\alpha_i, P_i) \mid i \leq m\}$  where  $\alpha_i$  is a parameter (weight or bias) of  $\mathcal{N}$  and  $P_i$  is a set of bit positions with  $|P_i| \leq n$ . We use  $\mathcal{N}^\rho$  to denote the resulting network by invoking the attack vector  $\rho$  on  $\mathcal{N}$ .

An  $(m, n)$ -attack vector defines the vulnerable parameters and bits that are flipped by the adversary during a specific BFA.

*Example 2.4.* Consider the QNN given in Example 2.1. Let  $m = n = 2$  and an attack vector  $\rho = \{(\widehat{\mathbf{W}}_{1,1}^2, \{2, 4\}), (\widehat{\mathbf{W}}_{1,2}^2, \{3\})\}$ . Intuitively,  $\rho$  describes a specific bit-flip attack that the 2nd and 4th bits in  $\vec{v}(\widehat{\mathbf{W}}_{1,1}^2) = [1001]$  and the 3rd bit in  $\vec{v}(\widehat{\mathbf{W}}_{1,2}^2) = [1101]$  are flipped. Then, we have the two's complement representations of attacked parameters as  $\vec{v}(\check{\mathbf{W}}_{1,1}^2) = [0011]$  and  $\vec{v}(\check{\mathbf{W}}_{1,2}^2) = [1001]$ .

Note that for clarity and convenience, given a QNN, the de-quantized parameters before (resp. after) BFAs  $\widehat{\mathbf{W}}_{j,k}^i$  (resp.  $\check{\mathbf{W}}_{j,k}^i$ ) may be directly represented by  $\mathbf{W}_{j,k}^i$  (resp.  $\check{\mathbf{W}}_{j,k}^i$ ) when it is clear from the context in the subsequent sections.

### 3 Bit-Flip Attack Verification Problem

In this section, we define the verification problem considered in this work and discuss a naive baseline solution based on DeepPoly.

#### 3.1 Problem Definition

*Definition 3.1 (BFA-tolerance).* Let  $\mathcal{N} : \mathbb{R}^n \rightarrow \mathbb{R}^s$  be a QNN. Given a pre-condition  $\phi$  over the input  $\mathbf{x} \in \mathbb{R}^n$  and post-condition  $\psi$  over the output  $\mathcal{N}(\mathbf{x}) \in \mathbb{R}^s$ . We use  $\mathcal{N} \models_{m,n}^\rho \langle \phi, \psi \rangle$  to denote that for any  $(m, n)$ -attack vector  $\rho$ ,  $\phi(\mathbf{x}) \Rightarrow \psi(\mathcal{N}^\rho(\mathbf{x}))$  always holds, where  $\mathcal{N}^\rho$  is the network obtained from  $\mathcal{N}$  given the attack vector  $\rho$ .

If  $\mathcal{N} \models_{m,n}^\rho \langle \phi, \psi \rangle$  holds, we say that  $\mathcal{N}$  is BFA-tolerant to the property  $\langle \psi, \phi \rangle$ . Note that, such a formulation of the problem is expressive enough to cover a range of desired neural network properties, including safety, robustness, (counterfactual) fairness, and backdoor-absence.

**THEOREM 3.2.** *Verifying whether  $\mathcal{N} \models_{m,n}^\rho \langle \phi, \psi \rangle$  holds is NP-complete.* □

In the following, for the sake of readability, our discussion focuses on the following general BFA-tolerant robustness property.

*Definition 3.3 (BFA-tolerant Robustness).* Let  $\mathcal{N} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be a QNN,  $\mathcal{I} \subset \mathbb{R}^n$  be an input region, and  $g$  is a target class.  $\mathcal{N}$  is BFA-tolerant for robustness with respect to the region  $\mathcal{I}$  and the class  $g$  if  $\mathcal{N} \models_{m,n}^\rho \langle \phi, \psi \rangle$  returns true, where  $\phi(\mathbf{x}) := \mathbf{x} \in \mathcal{I}$ ,  $\psi(\mathbf{y}) := \text{argmax}(\mathbf{y}) = g$ .



Intuitively, the BFA-tolerant robustness verification problem with  $m = n = 0$  is the vanilla robustness verification problem  $\mathcal{N} \models \langle \phi, \psi \rangle$  of neural networks, following the prior works [35]. In this work, we consider input regions that are expressible by polyhedra, following the literature, e.g., [21, 23, 29, 43, 45, 65, 66, 71, 72, 74, 81, 84, 88] to cite a few.

### 3.2 A Naive Method by DeepPoly

Next, we present a baseline approach that reduces the BFA verification problem to a classic neural network verification problem so that the existing verifier, such as DeepPoly [66], can be used to verify the above BFA-tolerant properties.

**Review of DeepPoly.** The key idea of DeepPoly is to approximate the behavior of the neural network based on an abstract interpreter specifically tailored to the setting of neural networks. Specifically, the abstract domain  $\mathcal{A}$  is a combination of polyhedra, coupled with abstract transformers for neural network functions, including affine functions and activation functions. To achieve this, each neuron in the hidden layer  $\mathbf{x}_j^i$  (the  $j$ -th neuron in the  $i$ -th layer) with  $\mathbf{x}_j^i = \text{ReLU}(\mathbf{W}^i \mathbf{x}^{i-1} + \mathbf{b}^i)$  is seen into two nodes  $\mathbf{x}_{j,1}^i$  and  $\mathbf{x}_{j,2}^i$  such that  $\mathbf{x}_{j,1}^i = \mathbf{W}_{j,:}^i \mathbf{x}^{i-1} + \mathbf{b}_j^i$  and  $\mathbf{x}_{j,2}^i = \text{ReLU}(\mathbf{x}_{j,1}^i)$ , where  $\mathbf{x}_{k,2}^{i-1} = \mathbf{x}_{k,2}^i$  for  $k \in [n_{i-1}]$ . Formally, the abstract element  $\mathbf{a}_{j,s}^i \in \mathcal{A}$  for each neuron  $\mathbf{x}_{j,s}^i$  ( $s \in \{1, 2\}$ ) is a tuple  $\langle a_{j,s}^{i,\leq}, a_{j,s}^{i,\geq}, l_{j,s}^i, u_{j,s}^i \rangle$ , where  $a_{j,s}^{i,\leq}$  (resp.  $a_{j,s}^{i,\geq}$ ) is a symbolic lower (resp. upper) bound in the form of a linear combination of variables which appear before it and  $l_{j,s}^i, u_{j,s}^i \in \mathbb{R}$ . For an affine function  $\mathbf{x}_{j,1}^i = \mathbf{W}_{j,:}^i \mathbf{x}^{i-1} + \mathbf{b}_j^i$ , the abstract transformer sets  $a_{j,1}^{i,\leq} = a_{j,1}^{i,\geq} = \mathbf{W}_{j,:}^i \mathbf{x}^{i-1} + \mathbf{b}_j^i$ . To compute the concrete lower (resp. upper) bound  $l_{j,1}^i$  (resp.  $u_{j,1}^i$ ), we first repeatedly substitute the variables in  $a_{j,1}^{i,\leq}$  (resp.  $a_{j,1}^{i,\geq}$ ) with their symbolic bounds according to the coefficient until no further substitution is possible. Then, we can obtain a sound lower (resp. upper) bound in the form of the linear combination of input variables, and  $l_{j,1}^i$  (resp.  $u_{j,1}^i$ ) can be computed immediately from the input domain. For an activation function  $\mathbf{x}_{j,2}^i = \text{ReLU}(\mathbf{x}_{j,1}^i)$ , the abstract transformers set the abstract element  $\mathbf{a}_{j,2}^i = \langle a_{j,2}^{i,\leq}, a_{j,2}^{i,\geq}, l_{j,2}^i, u_{j,2}^i \rangle$  as follows:

- If  $l_{j,1}^i \geq 0$ :  $a_{j,2}^{i,\leq} = a_{j,1}^{i,\leq}$ ,  $a_{j,2}^{i,\geq} = a_{j,1}^{i,\geq}$ ,  $l_{j,2}^i = l_{j,1}^i$ , and  $u_{j,2}^i = u_{j,1}^i$ ;
- If  $u_{j,1}^i \leq 0$ :  $a_{j,2}^{i,\leq} = a_{j,2}^{i,\geq} = l_{j,2}^i = u_{j,2}^i = 0$ ;
- If  $l_{j,1}^i < 0 < u_{j,1}^i$ :  $a_{j,2}^{i,\geq} = u_{j,1}^i (\mathbf{x}_{j,1}^i - l_{j,1}^i) / (u_{j,1}^i - l_{j,1}^i)$ ,  $a_{j,2}^{i,\leq} = \lambda \mathbf{x}_{j,1}^i$  where  $\lambda \in \{0, 1\}$  such that the area of resulting shape by  $a_{j,2}^{i,\leq}$  and  $a_{j,2}^{i,\geq}$  is minimal,  $l_{j,2}^i = \lambda l_{j,1}^i$  and  $u_{j,2}^i = u_{j,1}^i$ .

**A naive method.** Given the problem of verifying whether  $\mathcal{N} \models_{m,n}^\rho \langle \phi, \psi \rangle$  holds, a naive solution is to iteratively create an attacked network  $\mathcal{N}^\rho$  for each possible  $(m, n)$ -attack vector  $\rho$  and check the vanilla robustness verification problem  $\mathcal{N}^\rho \models \langle \phi, \psi \rangle$  by DeepPoly, which conducts a reachability analysis and returns a sound and incomplete verification result. Following this method, the number of possible attack vectors increases quickly with  $m, n$ , and the number of parameters in  $\mathcal{N}$ , causing the infamous combinatorial explosion problem. For instance, suppose the number of parameters of a QNN is  $K$  and the quantization bit-width is  $Q$ , the number of possible attack vectors (or the number of attacked networks  $\mathcal{N}^\rho$ ) is  $\binom{K}{m} \times (\sum_{i=1}^n \binom{Q}{i})^m$ .

## 4 Methodology of BFAVerifier

In this work, we operate under the assumption that the adversary is limited to attacking a small number of parameters in a QNN, specifically targeting only one parameter by default ( $m = 1$ ). Note that even if the adversary is limited to flipping only one parameter, the number of possible  $(1, n)$ -attack vectors is still  $K \cdot \sum_{i=1}^n \binom{Q}{i}$ . Consider a QNN  $\mathcal{N}$  which is quantized by  $Q$  and comprises  $K$  parameters. The naive method introduced in Section 3.1 can only verify each  $(1, n)$ -attack vector

**Algorithm 1:** Overall Algorithm of BFAVerifier

---

```

1 Proc BFA_Verifier( $\mathcal{N}, \mathcal{I}, g, n$ )
2    $\xi = \emptyset$ ;
3   foreach parameter  $w$  in  $\mathcal{N}$  do
4     if BFA_RA( $\mathcal{N}, \mathcal{I}, g, w, n$ ) = Unknown then
5        $\xi.append(w)$ ;
6   if  $\xi == \emptyset$  then
7     return True;
8   else
9     return BFA_MILP( $\mathcal{N}, \mathcal{I}, g, \xi, n$ );

```

---

separately and invokes  $K \cdot \sum_{i=1}^n \binom{Q}{i}$  times DeepPoly in total, which is highly inefficient. Our idea is to verify multiple attack vectors at one time.

#### 4.1 Overview of BFAVerifier

The overall verification procedure is given in Algorithm 1. Given a QNN  $\mathcal{N}$ , an input region  $\mathcal{I}$ , a target class  $g$ , and the maximum number of bits to flip  $n$ , we firstly traverse each parameter  $w$ , performing a reachability analysis via function BFA\_RA( $\cdot$ ) independently (lines 3-4) to compute a sound output range for  $\mathcal{N}$  considering all potential  $\sum_{i=1}^n \binom{Q}{i}$  attack vectors with respect to parameter  $w$ , and subsequently identify all parameters potentially susceptible to bit-flip attacks (line 5). If the set  $\xi$  is empty, we return True which means all parameters are safe to BFA and the network  $\mathcal{N}$  is BFA-tolerant with respect to the region  $\mathcal{I}$  and class  $g$ . Otherwise, it implies the existence of at least one parameter for which the reachability analysis fails to confirm safety against such attacks. In this case, we reformulate the verification problem into an equivalent MILP problem based on the intermediate results (i.e., all susceptible parameters  $\xi$ ) derived before, which can then be solved using off-the-shelf solvers. Therefore, the whole verification process BFAVerifier is sound, complete yet reasonably efficient. We remark that the MILP-based verification method is often more time-consuming and thus the first step allows us to quickly verify many tasks first or identify all vulnerable parameters soundly and formally.

Below, we present the details of functions BFA\_RA and BFA\_MILP. We first introduce an abstract domain designed for networks with symbolic parameters, which will be utilized throughout our reachability analysis procedure.

#### 4.2 SymPoly: An Abstract Domain for Networks with Symbolic Parameters

In this section, we introduce a new abstract domain SymPoly designed for networks with symbolic parameters, equipped with abstract transformers tailored to our bit-flip attack setting to conduct a sound reachability analysis.

Let us consider the  $(i+1)$ -th layer with neuron function  $\mathbf{x}_j^{i+1} = \text{ReLU}(\mathbf{W}_{j,:}^{i+1} \mathbf{x}^i + \mathbf{b}_j^{i+1})$  in a QNN  $\mathcal{N}$  such that  $\mathbf{W}_{j,k}^{i+1}$  (for some  $k \in [n_i]$ ) or/and  $\mathbf{b}_j^{i+1}$  may be replaced by symbolic parameters. Following DeepPoly, we first split each neuron (e.g.,  $\mathbf{x}_j^{i+1}$ ) into two nodes (e.g.,  $\mathbf{x}_{j,1}^{i+1}$  and  $\mathbf{x}_{j,2}^{i+1}$ ) and reformulate the neuron function as follows:

$$\mathbf{x}_{j,1}^{i+1} = \sum_{t \in [n_i]} \mathbf{W}_{j,t}^{i+1} \mathbf{x}_{t,2}^i + \mathbf{b}_j^{i+1}, \quad \mathbf{x}_{j,2}^{i+1} = \text{ReLU}(\mathbf{x}_{j,1}^{i+1})$$

**4.2.1 Abstract domain.** We inherit the abstract domain  $\mathcal{A}$  introduced in DeepPoly which consists of a set of polyhedral constraints, each relating one variable to a linear combination of the variables



from preceding layers. Formally, the abstract element of each neuron  $\mathbf{x}_{j,s}^{i+1}$  ( $s \in \{1, 2\}$ ) in our abstract domain is represented as  $\mathbf{a}_{j,s}^{i+1} = \langle a_{j,s}^{i+1,\leq}, a_{j,s}^{i+1,\geq}, l_j^{i+1}, u_j^{i+1} \rangle \in \mathcal{A}$ , and it satisfy the following invariant:  $\gamma(\mathbf{a}_{j,s}^{i+1}) = \{x \in \mathbb{R} \mid a_{j,s}^{i+1,\leq} \leq x \leq a_{j,s}^{i+1,\geq}\} \subseteq [l_j^{i+1}, u_j^{i+1}]$ . By repeatedly substituting each variable  $\mathbf{x}_{j',s}^{i'}$  in  $a_{j,s}^{i+1,\leq}$  (resp.  $a_{j,s}^{i+1,\geq}$ ) using  $a_{j',s}^{i',\leq}$  or  $a_{j',s}^{i',\geq}$  according to the coefficient of  $\mathbf{x}_{j',s}^{i'}$ , until no further substitution is possible,  $a_{j,s}^{i+1,\leq}$  (resp.  $a_{j,s}^{i+1,\geq}$ ) will be a linear combination over the input variables of the QNN. We next introduce our abstract transformation for functions in  $\mathcal{N}$ .

**4.2.2 Affine abstract transformer for symbolic weights.** Without loss of generality, we consider the transformer for the case where there is only one concrete parameter is replaced by a symbolic one, e.g.,  $\vec{\mathbf{W}}_{j,k}^{i+1}$  for some  $k \in [n_i]$ . For all nodes other than  $\mathbf{x}_{j,1}^{i+1}$ , we directly inherit the abstract transformers from DeepPoly.

In this work, we need to abstract affine functions with symbolic parameters and the ReLU function, both of which contribute to precision loss. To improve accuracy, we abstract them jointly as a symbolic weighted ReLU function with ReLU applied internally, as shown in Figure 3b. For the very first affine layer, we abstract the affine function solely, since there is no preceding ReLU, as given later at the end of this section. We remark that our abstract transformations can be compositionally applied to settings involving multiple symbolic parameters.

**Symbolic weights on hidden neurons.** Consider a symbolic weight parameter  $\vec{\mathbf{W}}_{j,k}^{i+1}$  constrained by an interval range  $[w_l, w_u]$ . Then, the updated neuron function for  $\mathbf{x}_{j,1}^{i+1}$  is as follows:

$$\mathbf{x}_{j,1}^{i+1} = \sum_{t \in [n_i] \setminus k} \mathbf{W}_{j,t}^{i+1} \mathbf{x}_{t,2}^i + \vec{\mathbf{W}}_{j,k}^{i+1} \mathbf{x}_{k,2}^i + \mathbf{b}_j^{i+1} \quad (1)$$

To perform abstract transformations on  $\vec{\mathbf{W}}_{j,k}^{i+1} \mathbf{x}_{k,2}^i$ , an intuitive idea is to directly make an affine transformation with respect to symbolic parameter  $\vec{\mathbf{W}}_{j,k}^{i+1}$  on the abstract element of  $\mathbf{x}_{k,2}^i$ . However, it will lead to an over-approximate result compared to abstracting the symbolic-weighted ReLU function. To illustrate it, let us consider the setting where  $(l_{k,2}^i < 0 < u_{k,2}^i) \wedge (l_{k,2}^i + u_{k,2}^i > 0)$  and  $w_l \geq 0$ . As shown in Figure 3(a), the areas within the yellow boundaries and the green boundaries are captured by the weighted abstract elements  $w_u \cdot \mathbf{a}_{k,2}^i$  and  $w_l \cdot \mathbf{a}_{k,2}^i$ , respectively, with  $\gamma(w_u \cdot \mathbf{a}_{k,2}^i) = \{w_u \cdot x \in \mathbb{R} \mid a_{k,2}^{i,\leq} \leq x \leq a_{k,2}^{i,\geq}\}$  and  $\gamma(w_l \cdot \mathbf{a}_{k,2}^i) = \{w_l \cdot x \in \mathbb{R} \mid a_{k,2}^{i,\leq} \leq x \leq a_{k,2}^{i,\geq}\}$ . It is obvious that the area captured by the dotted polyhedra in Figure 3(a) is larger than that in Figure 3(b), whose area is captured by directly abstracting the weighted ReLU function  $\vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i)$ .

Therefore, our idea is to abstract the symbolic-weighted ReLU function directly. To achieve it, we initially introduce an additional node  $\tilde{\mathbf{x}}_{k,2}^i$  for the symbolic parameter  $\vec{\mathbf{W}}_{j,k}^{i+1}$  such that  $\tilde{\mathbf{x}}_{k,2}^i = \text{ReLU}_{w_l}(\mathbf{x}_{k,1}^i) = \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i)$ . After that, we set the weight as 1 between  $\tilde{\mathbf{x}}_{k,2}^i$  and  $\mathbf{x}_{j,1}^{i+1}$ , and set the weight between  $\mathbf{x}_{k,2}^i$  and  $\mathbf{x}_{j,1}^{i+1}$  as 0. An illustration of the construction can be found in Figure 4. Then, given the abstract element of  $\mathbf{x}_{k,2}^i$  as  $\mathbf{a}_{k,2}^i = \langle a_{k,2}^{i,\leq}, a_{k,2}^{i,\geq}, l_{k,2}^i, u_{k,2}^i \rangle$ , we define the abstract element  $\mathbf{a}_{k,2}^{i,*} = \langle \tilde{a}_{k,2}^{i,\leq}, \tilde{a}_{k,2}^{i,\geq}, \tilde{l}_{k,2}^i, \tilde{u}_{k,2}^i \rangle$  of neuron  $\tilde{\mathbf{x}}_{k,2}^i$  as follows:

- If  $w_l \geq 0$ :  $\tilde{a}_{k,2}^{i,\leq} = w_l \cdot a_{k,2}^{i,\leq}$ ,  $\tilde{a}_{k,2}^{i,\geq} = w_u \cdot a_{k,2}^{i,\geq}$ ,  $\tilde{l}_{k,2}^i = w_l \cdot l_{k,2}^i$ , and  $\tilde{u}_{k,2}^i = w_u \cdot u_{k,2}^i$ ;
- If  $w_u \leq 0$ :  $\tilde{a}_{k,2}^{i,\leq} = w_l \cdot a_{k,2}^{i,\leq}$ ,  $\tilde{a}_{k,2}^{i,\geq} = w_u \cdot a_{k,2}^{i,\geq}$ ,  $\tilde{l}_{k,2}^i = w_l \cdot u_{k,2}^i$ , and  $\tilde{u}_{k,2}^i = w_u \cdot l_{k,2}^i$ ;
- If  $w_l < 0 < w_u$ :  $\tilde{a}_{k,2}^{i,\leq} = w_l \cdot a_{k,2}^{i,\leq}$ ,  $\tilde{a}_{k,2}^{i,\geq} = w_u \cdot a_{k,2}^{i,\geq}$ ,  $\tilde{l}_{k,2}^i = w_l \cdot u_{k,2}^i$ , and  $\tilde{u}_{k,2}^i = w_u \cdot u_{k,2}^i$ .

An illustration of the above abstract transformer for the weighted-ReLU function can be found in Figure 5.

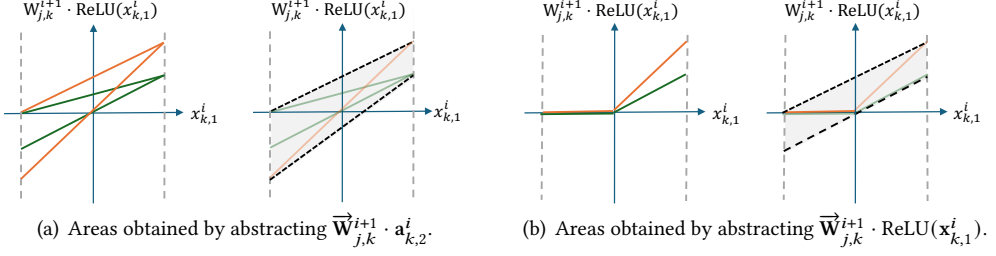


Fig. 3. Convex approximations of  $\vec{W}_{j,k}^{i+1} x_{k,2}^i = \vec{W}_{j,k}^{i+1} \cdot \text{ReLU}(x_{k,1}^i)$  via different abstract transformations: (a) depicts the approximation derived by abstracting  $\vec{W}_{j,k}^{i+1} \cdot \mathbf{a}_{k,2}^i$ , where  $\mathbf{a}_{k,2}^i$  is an over-approximation of  $\text{ReLU}(x_{k,1}^i)$  obtained from DeepPoly, and the yellow and green lines gives the boundaries of  $w_u \cdot \mathbf{a}_{k,2}^i$  and  $w_l \cdot \mathbf{a}_{k,2}^i$ . (b) directly give the approximation by linear boundaries with the minimal area in the input-output plane of the function  $\vec{W}_{j,k}^{i+1} \cdot \text{ReLU}(x_{k,1}^i)$ .

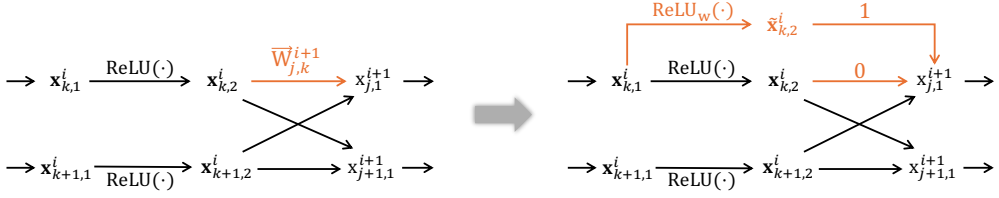


Fig. 4. The construction of additional node  $\tilde{x}_{k,2}^i$  given a symbolic parameter  $\vec{W}_{j,k}^{i+1}$ .

**THEOREM 4.1.** *The weighted ReLU abstract transformer for neuron  $\tilde{x}_{k,2}^i$  in Figure 4 i) is sound and preserves the invariant  $\gamma(\mathbf{a}_{k,2}^{i,*}) \subseteq [\tilde{l}_{k,2}^i, \tilde{u}_{k,2}^i]$ .  $\square$*

Finally, we can apply the affine abstract transformers introduced in DeepPoly (cf. Section 3.2) to the remaining affine transformations within  $\mathbf{x}_{j,1}^{i+1}$  in equation (1), i.e.,  $\sum_{t \in [n_i] \setminus k} \mathbf{W}_{j,t}^{i+1} \mathbf{x}_{t,2}^i + \mathbf{b}_j^{i+1}$ . Hence, both soundness and the domain invariant are preserved in our affine abstract transformers on equation (1) considering symbolic weight parameters.

**Symbolic weights on input neurons.** Consider  $\mathbf{x}_{j,1}^2 = \sum_{t \in [n_2] \setminus k} \mathbf{W}_{j,t}^2 \mathbf{x}_t^1 + \vec{W}_{j,k}^2 \mathbf{x}_k^1 + \mathbf{b}_j^2$  with a symbolic weight  $\vec{W}_{j,k}^2$  connected to the input neuron  $\mathbf{x}_k^1$ . Let  $[x_l, x_u]$  be the input range of the neuron  $\mathbf{x}_k^1$ , then the abstract domain for  $\mathbf{x}_{j,1}^2$  is  $\mathbf{a}_{j,1}^{2,*} = \langle \tilde{a}_{j,1}^{2,\leq}, \tilde{a}_{j,1}^{2,\geq}, \tilde{l}_{j,1}^2, \tilde{u}_{j,1}^2 \rangle$  for  $\mathbf{x}_{j,1}^2 = \sum_{t \in [n_2] \setminus k} \mathbf{W}_{j,t}^2 \mathbf{x}_t^1 + \vec{W}_{j,k}^2 \mathbf{x}_k^1 + \mathbf{b}_j^2$  with  $\tilde{a}_{j,1}^{2,\leq}$  and  $\tilde{a}_{j,1}^{2,\geq}$  set as follows:

$$\tilde{a}_{j,1}^{2,\leq} = \sum_{t \in [n_2] \setminus k} \mathbf{W}_{j,t}^2 \mathbf{x}_t^1 + \mathbf{b}_j^2 + \kappa^{\leq} \mathbf{x}_k^1 - \eta, \quad \tilde{a}_{j,1}^{2,\geq} = \sum_{t \in [n_2] \setminus k} \mathbf{W}_{j,t}^2 \mathbf{x}_t^1 + \mathbf{b}_j^2 + \kappa^{\geq} \mathbf{x}_k^1 + \eta$$

where if  $0 \leq x_l$ , then  $\{\kappa^{\leq} = w_l, \kappa^{\geq} = w_u, \eta = 0\}$ ; if  $x_u \leq 0$ , then  $\{\kappa^{\leq} = w_u, \kappa^{\geq} = w_l, \eta = 0\}$ ; Otherwise,  $\{\kappa^{\leq} = \frac{w_l x_u - w_u x_l}{x_u - x_l}, \kappa^{\geq} = \frac{w_u x_u - w_l x_l}{x_u - x_l}, \eta = \frac{x_u x_l}{x_u - x_l} (w_l - w_u)\}$ .

$\tilde{l}_{j,1}^2$  and  $\tilde{u}_{j,1}^2$  can be determined with corresponding lower/upper bounds computation methods (cf. Section 3.2). Intuitively,  $\kappa^{\leq} \mathbf{x}_k^1 - \eta$  (resp.  $\kappa^{\geq} \mathbf{x}_k^1 + \eta$ ) expresses the lower (resp. upper) boundary of the abstract domain of the weighted input neuron  $\vec{W}_{j,k}^2 \mathbf{x}_k^1$ .

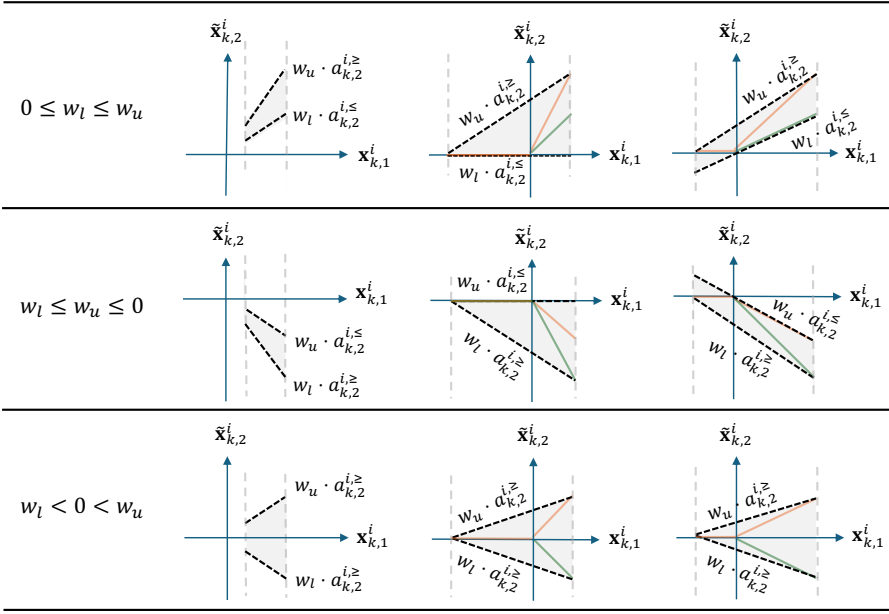


Fig. 5. An illustration of the abstract transformer for the symbolic-weighted ReLU function  $\vec{W}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i)$ , where the first column defines the range (interval) of values  $[w_l, w_u]$  for  $\vec{W}_{j,k}^{i+1}$ . The second column gives the abstraction when  $l_{k,1}^i > 0$  for the abstract element  $a_{k,1}^i$  of  $\mathbf{x}_{k,1}^i$ , and the third (resp. fourth) column shows the abstraction of when  $(l_{k,1}^i < 0 < u_{k,1}^i) \wedge (l_{k,1}^i + u_{k,1}^i \leq 0)$  (resp.  $(l_{k,1}^i < 0 < u_{k,1}^i) \wedge (l_{k,1}^i + u_{k,1}^i > 0)$ ).

**THEOREM 4.2.** *The abstract transformer for symbolic weighted input neuron  $\mathbf{x}_{j,2}^2$  is sound and preserves the invariant  $\gamma(\mathbf{a}_{j,1}^{2,*}) \subseteq [\tilde{l}_{j,1}^2, \tilde{u}_{j,1}^2]$ .*  $\square$

**4.2.3 Affine abstract transformer for symbolic biases.** Similar to the affine transformer for symbolic weights, for all nodes other than  $\mathbf{x}_{j,1}^{i+1}$ , we adopt the abstract transformers from DeepPoly. Our abstract transformations can be compositionally applied to settings involving multiple symbolic parameters.

Consider a symbolic bias parameter  $\vec{\mathbf{b}}_j^{i+1}$  constrained by an interval range  $[w_l, w_u]$ . Then, the updated neuron function is  $\mathbf{x}_{j,1}^{i+1} = \sum_{t \in [n_i]} \mathbf{W}_{j,t}^{i+1} \mathbf{x}_{t,2}^i + \vec{\mathbf{b}}_j^{i+1}$ . Then, we define the abstract element  $\mathbf{a}_{j,1}^{i+1} = \langle a_{j,1}^{i+1,\leq}, a_{j,1}^{i+1,\geq}, l_{j,1}^{i+1}, u_{j,1}^{i+1} \rangle$  of neuron  $\mathbf{x}_{j,1}^{i+1}$  as follows:

$$a_{j,1}^{i+1,\leq} = \sum_{t \in [n_i]} \mathbf{W}_{j,t}^{i+1} \mathbf{x}_{t,2}^i + w_l, \quad a_{j,1}^{i+1,\geq} = \sum_{t \in [n_i]} \mathbf{W}_{j,t}^{i+1} \mathbf{x}_{t,2}^i + w_u$$

where  $l_{j,1}^{i+1}$  and  $u_{j,1}^{i+1}$  can be determined with corresponding lower/upper bounds computation methods (cf. Section 3.2).

**THEOREM 4.3.** *The affine abstract transformer for symbolic biases preserves both soundness and the invariant.*  $\square$

**Other abstract transformers.** In this work, for other network functions, such as the ReLU function and the maxpool operator, we directly adopt the corresponding abstract transformers from DeepPoly. Hence, SymPoly is sound.

### 4.3 Details of Function BFA\_RA

In this section, we give the implementation details of our reachability analysis procedure BFA\_RA. For any parameter  $w$ , we can always determine its concrete interval  $[w_l, w_u]$  where any value obtained by flipping up to  $n$  bits will invariably be contained. Based on this interval, given a QNN  $\mathcal{N}$ , an input region  $\mathcal{I}$ , and a target class  $g$ , we can use SymPoly to perform reachability analysis on a modified network  $\mathcal{N}^*$ , where the concrete parameter  $w$  is substituted with a symbolic parameter  $\vec{w}$  constrained by the interval range  $[w_l, w_u]$ . Then, the analysis will yield two results: i) *Proved*, indicating that the adversary cannot compromise  $\mathcal{N}$  (e.g., all inputs from the input region  $\mathcal{I}$  will be classified as the same class  $g$ ) by altering at most  $n$  bits in the two's complement representation of the parameter  $w$ ; or ii) *Unknown*, meaning that we cannot confirm the security of  $w$  against such bit-flip attacks and there may exist some inputs within the region  $\mathcal{I}$  that will be misclassified by the attacked network into classes other than  $g$ .

We remark that not all bits of a parameter are equally important concerning BFAs. For example, given a quantized parameter  $[v_Q; v_{Q-1}; \dots; v_1]$ , flipping the signal bit  $v_Q$  would always produce the most deviation than flipping any other bit  $v_i$  (for  $i \in [Q-1]$ ) since the former always results in the largest parameter interval by adding or subtracting  $2^{Q-1}$  onto the original parameter value. Hence, if we consider all the bits at one time, we need to verify the neural network with a large interval for the parameter, and likely return *Unknown*. Moreover, according to the abstract transformers defined for symbolic weight parameters in Section 4.2.2, when the weight interval  $[w_l, w_u]$  satisfies  $w_l < 0 < w_u$ , the abstract transformer for the  $\text{ReLU}_w$  function would lead to a looser over-approximation compared to the other two settings ( $w_l \geq 0$  or  $w_u \leq 0$ ). Hence, given a symbolic parameter with a constrained interval range  $[w_l, w_u]$ , to enhance the precision of our reachability analysis result, we first partition the parameter interval into two sub-intervals characterized by uniformly signed parameter values, either entirely positive or entirely negative. Then, we perform reachability analysis using SymPoly separately for each sub-interval. Note that, this division addresses the significant over-approximation precision loss that occurs when a symbolic weight parameter has a lower bound and upper bound with differing signs. An example illustrating how such an interval partition enhances the abstraction precision is given in Appendix C. Moreover, for parameter intervals that are too wide to be proved by SymPoly, we introduce a binary search method, which splits the parameter interval at its midpoint and independently verifies each resulting smaller interval iteratively.

The details of function BFA\_RA can be found in Algorithm 2, where  $\text{SymPoly}(\mathcal{N}, \mathcal{I}, g, w, w_l, w_u)$  represents that we conduct reachability analysis via SymPoly on the network  $\mathcal{N}$  equipped with a symbolic parameter  $w$  constrained by the interval  $[w_l, w_u]$  with respect to the input region  $\mathcal{I}$  and output class  $g$ . The algorithm works as follows. Given a neural network  $\mathcal{N}$ , an input region  $\mathcal{I}$ , a target class  $g$ , an attacked parameter  $w$ , and the bit flipping maximum number  $n$ , we first compute two intervals  $[w_l^+, w_u^+]$  and  $[w_l^-, w_u^-]$  for  $w$  such that i)  $w_l^+ \geq 0$  and  $w_u^- \leq 0$ , and ii) these intervals are designed to encompass any values obtained by flipping up to  $n$  bits in the two's complement representation of  $w$ , which can be done as follows:

- If  $w \geq 0$ :  $w_l^+$  (resp.  $w_u^+$ ) is obtained by flipping the most significant  $n$  bits via  $1 \rightarrow 0$  (resp.  $0 \rightarrow 1$ ), and  $w_l^-$  (resp.  $w_u^-$ ) is obtained by flipping the signal bit and the most significant  $n-1$  bits via  $1 \rightarrow 0$  (resp.  $0 \rightarrow 1$ );
- If  $w < 0$ :  $w_l^+$  (resp.  $w_u^+$ ) is obtained by flipping the signal bits via  $1 \rightarrow 0$  and the most significant  $n-1$  bits via  $1 \rightarrow 0$  (resp.  $0 \rightarrow 1$ ), and  $w_l^-$  (resp.  $w_u^-$ ) is obtained by the most significant  $n$  bits via  $1 \rightarrow 0$  (resp.  $0 \rightarrow 1$ ).

For parameter intervals that are too wide to be verified by SymPoly (cf. line 13), we introduce a binary search method, which splits the parameter interval at its midpoint and independently

**Algorithm 2: BFA\_RA function**


---

```

1 Proc BFA_RA( $\mathcal{N}, \mathcal{I}, g, w, n$ )
2   Compute two minimal sub-intervals  $[w_l^+, w_u^+], [w_l^-, w_u^-]$  for  $w$  such that i)  $w_l^+ \geq 0 \wedge w_u^- \leq 0$ , and ii)
    $[w_l^+, w_u^+] \cup [w_l^-, w_u^-]$  consists of all possible values of  $w$  that can be derived by flipped at most  $n$  bits;
3   if binary_RA( $\mathcal{N}, \mathcal{I}, g, w, w_l^+, w_u^+$ ) = Unknown then
4     | return Unknown;
5   if binary_RA( $\mathcal{N}, \mathcal{I}, g, w, w_l^-, w_u^-$ ) = Unknown then
6     | return Unknown;
7   return Proved;

8 Proc binary_RA( $\mathcal{N}, \mathcal{I}, g, w, w_l, w_u$ )
9   if SymPoly( $\mathcal{N}, \mathcal{I}, g, w, w_l, w_u$ ) = Proved then
10    | return Proved;
11  if  $w_l == w_u$  then
12    | return SymPoly( $\mathcal{N}, \mathcal{I}, g, w, w_l, w_u$ );
13  Split  $[w_l, w_u]$  at the midpoint and get two minimal sub-intervals  $[w_l, w'_l], [w'_l, w_u]$  such that
    $[w_l, w'_l] \cup [w'_l, w_u]$  encompasses all potential flipped values of  $w$  as that in  $[w_l, w_u]$ ;
14  if binary_RA( $\mathcal{N}, \mathcal{I}, g, w, w_l, w'_l$ ) = Unknown then
15    | return Unknown;
16  else if binary_RA( $\mathcal{N}, \mathcal{I}, g, w, w'_l, w_u$ ) = Unknown then
17    | return Unknown;
18  else
19    | return Proved;

```

---

verifies each resultant smaller interval iteratively. Smaller intervals are generally more likely to yield Proved results, thus enhancing the overall effectiveness and precision of our reachability analysis, which has been confirmed by our experiments (cf. Section 5.1).

#### 4.4 Details of Function BFA\_MILP

If BFA\_RA fails to prove the BFA-tolerant robustness property, we then encode the verification problem as an equivalent MILP problem w.r.t the set of unproved parameters as follows.

**Encoding of input regions.** We consider input regions that are expressible by polyhedra in this work, and they can be directly encoded by linear constraints. For example, for an input region defined by  $L_\infty$ -norm  $\mathcal{I}_u^r = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{u}\|_\infty \leq r\}$ , we can use the following constraint set  $\Theta^{\mathcal{I}}$  to encode the input condition  $\phi(\mathbf{x}) := \mathbf{x} \in \mathcal{I}_u^r$ :

$$\Theta^{\mathcal{I}} = \{\max(\mathbf{u}_i - r, 0) \leq \mathbf{x}_i \leq \min(\mathbf{u}_i + r, 1) \mid i \in [n]\}$$

For input regions defined by the cartesian product of intervals  $\mathcal{I}^{1 \times u} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}_i \in [l_i, u_i]\}$ , we can use the following constraint set to encode the input condition  $\phi(\mathbf{x}) := \mathbf{x} \in \mathcal{I}^{1 \times u}$ :

$$\Theta^{\mathcal{I}} = \{l_i \leq \mathbf{x}_i \leq u_i \mid i \in [n]\}$$

**Encoding of output properties.** Let  $\mathbf{y}$  denote the output vector of  $\mathcal{N}^\rho$  given any attack vector  $\rho$ . We encode the output condition, i.e.,  $\psi(\mathbf{y}) := \operatorname{argmax}(\mathbf{y}) \neq \operatorname{argmax}(\mathcal{N}(\mathbf{u})) = g \in [s]$  into the following set of constraints based on a set of Boolean variables  $\{\eta_i \mid i \in [s] \setminus g\}$ :

- If  $i < g$ : then  $\mathbf{y}_i \geq \mathbf{y}_g \Leftrightarrow \eta_i = 1$  which can be encoded as  $\Theta_{i,0}^g = \{\mathbf{y}_g + \mathbf{M} \cdot (\eta_i - 1) \leq \mathbf{y}_i, \mathbf{y}_i < \mathbf{y}_g + \mathbf{M} \cdot \eta_i\}$ ;
- If  $i > g$ : then  $\mathbf{y}_i > \mathbf{y}_g \Leftrightarrow \eta_i = 1$  which can be encoded as  $\Theta_{i,1}^g = \{\mathbf{y}_g + \mathbf{M} \cdot (\eta_i - 1) < \mathbf{y}_i, \mathbf{y}_i < \mathbf{y}_g + \mathbf{M} \cdot \eta_i\}$ .

Intuitively,  $\Theta_{i,0}^g$  (resp.  $\Theta_{i,1}^g$ ) ensures that the  $i$ -th entry of the output vector  $\mathbf{y}$  for  $i < g$  (resp.  $i > g$ ) is no less than (resp. larger than) the  $g$ -th entry iff  $\eta_i = 1$ . As a result,  $\text{argmax}(\mathbf{y}) \neq g$  iff the set of constraints  $\Theta_g = \bigcup_{i < g} \Theta_{i,0}^g \cup \bigcup_{i > g} \Theta_{i,1}^g \cup \{\sum_{i \in [s] \setminus g} \eta_i \geq 1\}$  holds.

**Encoding of neural networks under BFAs.** Next, we present the MILP encoding of the neural network under BFAs based on the intermediate analysis results from function BFA\_RA.

Let  $\xi = \{w_1, w_2, \dots, w_m\}$  denote the vulnerable parameters set obtained from our reachability analysis (cf. Algorithm 1). Each parameter is quantized by  $Q$  bits. The adversary can only attack one parameter from the parameter set  $\xi$  and flip  $\mathfrak{n}$  bit at most on it. We first traverse all possible  $(1, \mathfrak{n})$ -attack vectors for each parameter  $w_i \in \xi$  ( $1 \leq i \leq |\xi|$ ), and get the flipped value set  $F_{w_i} = \{w_i^1, w_i^2, \dots, w_i^{\mathfrak{R}}\}$  for each  $w_i$ , where  $\mathfrak{R} = \sum_{i=1}^{\mathfrak{n}} \binom{Q}{i}$ . Then, we use the following constraint set  $\Theta_{\xi}^{\mathfrak{n}}$  to encode the parameters in  $\xi$ , where  $\delta_i^j$  for  $i \in |\xi|$  and  $j \in \mathfrak{R}$  are binary variables:

$$\Theta_{\xi}^{\mathfrak{n}} = \left\{ \begin{array}{l} \tilde{w}_i = w_i + (w_i^1 - w_i)\delta_i^1 + (w_i^2 - w_i)\delta_i^2 + \dots + (w_i^{\mathfrak{R}} - w_i)\delta_i^{\mathfrak{R}}, \\ i \in |\xi|, \quad \sum_{i=1}^{|\xi|} \sum_{j=1}^{\mathfrak{R}} \delta_i^j = 1 \end{array} \right\}$$

Intuitively, for each parameter  $w_i$ , the binary variable  $\delta_i^j = 1$  indicates that the adversary attacks the parameter  $w_i$  and alters it into a new value  $w_i^j$ . If  $\sum_{j=1}^{\mathfrak{R}} \delta_i^j = 0$ , then it means that the adversary does not attack the parameter  $w_i$ . The constraint  $\sum_{i=1}^{|\xi|} \sum_{j=1}^{\mathfrak{R}} \delta_i^j = 1$  ensures that only one parameter is altered, while no more than  $\mathfrak{n}$  bits are subject to modification.

Then, we follow the existing MILP encoding method [51] to encode  $\mathcal{N}^{\rho}$  into a set of mixed-integer linear constraints  $\Theta_{\mathcal{N}^{\rho}}$ , where for each vulnerable parameter  $w_i \in \xi$ , we use  $\tilde{w}_i$  in  $\Theta_{\xi}^{\mathfrak{n}}$  to replace  $w_i$  in the encoding of the affine function. Finally, the BFA-tolerant robustness verification problem is equivalent to the solving of the constraint set:  $\Theta_P = \Theta_{\mathcal{N}^{\rho}} \cup \Theta_{\xi}^{\mathfrak{n}} \cup \Theta^I \cup \Theta_g$ .

**THEOREM 4.4.**  $\mathcal{N} \models_{\mathfrak{m}, \mathfrak{n}}^{\rho}$  holds iff  $\Theta_P$  is unsatisfiable. □

Overall, the complexity of BFA\_RA is polynomial in the network size when  $\mathfrak{m} = 1$ , whereas BFA\_MILP remains NP-complete even when  $\mathfrak{m} = 1$ .

## 4.5 Extension to Other Networks

This work primarily focuses on feedforward neural networks with ReLU activations. In this section, we demonstrate the extensibility of our framework to other networks, including those with sigmoid or tanh activations and architectures incorporating convolutional layers.

**4.5.1 Other activation functions.** Following the idea of symbolic weights on hidden neurons in Section 4.2.2 and the abstract transformers proposed in DeepPoly for sigmoid and tanh, for an activation function  $g(x)$  that is continuous and twice-differentiable such that the first derivative  $g'(x) > 0$  and the second derivative  $g'' \geq 0 \Leftrightarrow x \leq 0$ , we also construct an additional node  $\tilde{\mathbf{x}}_{k,2}^i$  (the same as in Figure 4) and study its abstract domain according to  $\tilde{\mathbf{x}}_{k,2}^i = \vec{\mathbf{W}}_{j,k}^{i+1} \cdot g(\mathbf{x}_{k,1}^i)$ . The corresponding abstract transformers for Sigmoid and Tanh considering the node  $\tilde{\mathbf{x}}_{k,2}^i$  are given in Table 1. For the other network functions with constant parameters, we can reuse the corresponding abstract transformers from DeepPoly directly.

**THEOREM 4.5.** Both the weighted Sigmoid and the weighted Tanh abstract transformers are sound and preserve the invariant  $\gamma(\mathbf{a}_{k,2}^{i,*}) \subseteq [\tilde{l}_{k,2}^i, \tilde{u}_{k,2}^i]$ . □

For the MILP encoding of other activation functions, the piecewise linear approximation can be employed to encode the sigmoid and tanh functions using linear constraints. We argue that such an approximation-based MILP encoding approach is sound, however, not incomplete. Therefore, for



Table 1. The abstract domain  $\mathbf{a}_{k,2}^{i,*} = \langle \tilde{a}_{k,2}^{i,\leq}, \tilde{a}_{k,2}^{i,\geq}, \tilde{l}_{k,2}^i, \tilde{u}_{k,2}^i \rangle$  of  $\tilde{\mathbf{x}}_{k,2}^i = \vec{\mathbf{W}}_{j,k}^{i+1} \cdot g(\mathbf{x}_{k,1}^i)$ , where  $l_{k,1}^i$  and  $u_{k,1}^i$  are the lower and upper bounds of  $\mathbf{x}_{k,1}^i$ ,  $\kappa = \frac{g(u_{k,1}^i) - g(l_{k,1}^i)}{u_{k,1}^i - l_{k,1}^i}$ , and  $\kappa' = \min(g'(l_{k,1}^i), g'(u_{k,1}^i))$

$g(x)$	Bounds of $\mathbf{x}_{k,1}^i$	$w_l \geq 0$	$w_u \leq 0$
Sigmoid(x)	$l_{k,1}^i \geq 0$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(l_{k,1}^i) + w_l \kappa (\mathbf{x}_{k,1}^i - l_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(u_{k,1}^i) + w_l \kappa' (\mathbf{x}_{k,1}^i - u_{k,1}^i)$
		$\tilde{a}_{k,2}^{i,\geq} = w_u g(u_{k,1}^i) + w_u \kappa' (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\geq} = w_u g(l_{k,1}^i) + w_u \kappa (\mathbf{x}_{k,1}^i - l_{k,1}^i)$
		$\tilde{l}_{k,2}^i = w_l g(l_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(u_{k,1}^i)$	$\tilde{l}_{k,2}^i = w_l g(u_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(l_{k,1}^i)$
	$u_{k,1}^i \leq 0$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(l_{k,1}^i) + w_l \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(u_{k,1}^i) + w_l \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$
		$\tilde{a}_{k,2}^{i,\geq} = w_u g(u_{k,1}^i) + w_u \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\geq} = w_u g(l_{k,1}^i) + w_u \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$
		$\tilde{l}_{k,2}^i = w_l g(l_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(u_{k,1}^i)$	$\tilde{l}_{k,2}^i = w_l g(u_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(l_{k,1}^i)$
$l_{k,1}^i < 0 < u_{k,1}^i$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(l_{k,1}^i) + w_l \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(u_{k,1}^i) + w_l \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	
$\tilde{a}_{k,2}^{i,\geq} = w_u g(u_{k,1}^i) + w_u \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\geq} = w_u g(l_{k,1}^i) + w_u \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$		
$\tilde{l}_{k,2}^i = w_l g(l_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(u_{k,1}^i)$	$\tilde{l}_{k,2}^i = w_l g(u_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(l_{k,1}^i)$		
Tanh(x)	$l_{k,1}^i \geq 0$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(l_{k,1}^i) + w_l \kappa (\mathbf{x}_{k,1}^i - l_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(u_{k,1}^i) + w_l \kappa' (\mathbf{x}_{k,1}^i - u_{k,1}^i)$
		$\tilde{a}_{k,2}^{i,\geq} = w_u g(u_{k,1}^i) + w_u \kappa' (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\geq} = w_u g(l_{k,1}^i) + w_u \kappa (\mathbf{x}_{k,1}^i - l_{k,1}^i)$
		$\tilde{l}_{k,2}^i = w_l g(l_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(u_{k,1}^i)$	$\tilde{l}_{k,2}^i = w_l g(u_{k,1}^i), \tilde{u}_{k,2}^i = w_u g(l_{k,1}^i)$
	$u_{k,1}^i \leq 0$	$\tilde{a}_{k,2}^{i,\leq} = w_u g(l_{k,1}^i) + w_u \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\leq} = w_u g(u_{k,1}^i) + w_u \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$
		$\tilde{a}_{k,2}^{i,\geq} = w_l g(u_{k,1}^i) + w_l \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\geq} = w_l g(l_{k,1}^i) + w_l \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$
		$\tilde{l}_{k,2}^i = w_u g(l_{k,1}^i), \tilde{u}_{k,2}^i = w_l g(u_{k,1}^i)$	$\tilde{l}_{k,2}^i = w_u g(u_{k,1}^i), \tilde{u}_{k,2}^i = w_l g(l_{k,1}^i)$
$l_{k,1}^i < 0 < u_{k,1}^i$	$\tilde{a}_{k,2}^{i,\leq} = w_u g(l_{k,1}^i) + w_l \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\leq} = w_l g(u_{k,1}^i) + w_u \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	
$\tilde{a}_{k,2}^{i,\geq} = w_u g(u_{k,1}^i) + w_l \kappa (\mathbf{x}_{k,1}^i - u_{k,1}^i)$	$\tilde{a}_{k,2}^{i,\geq} = w_l g(l_{k,1}^i) + w_u \kappa' (\mathbf{x}_{k,1}^i - l_{k,1}^i)$		
$\tilde{l}_{k,2}^i = w_u g(l_{k,1}^i), \tilde{u}_{k,2}^i = w_l g(u_{k,1}^i)$	$\tilde{l}_{k,2}^i = w_l g(u_{k,1}^i), \tilde{u}_{k,2}^i = w_l g(l_{k,1}^i)$		

these logistic activation functions, we can only claim that our approach is sound but incomplete, and limit our focus to the BFA\_RA component.

**4.5.2 Other network architectures.** This work focuses on feed-forward network architectures, however, our approach can be generalized to shared-parameter architectures—such as convolutional networks—without additional technical challenges. Figure 6 illustrates how a convolutional layer subjected to bit-flip attacks on the parameter  $w$  can be transformed into an equivalent affine layer, to which BFAVerifier can be directly applied. Note that, in Figure 6(b), although multiple copies are under bit-flip attack, they share the same parameter  $w$  in the original convolutional layer, and consequently, the attack effect is identical. Therefore, no additional combinatorial explosion occurs and the computational complexity remains equivalent to the case when  $m = 1$ .

Although no additional technical challenges occurs, BFAVerifier may suffer from significant abstraction precision loss on CNNs, compared to DNNs, due to multiple abstractions in weighted activation function and input neurons even when  $m = 1$  (in contrast to only single abstraction in feedforward networks), both contributing to higher loss than in cases without symbolic weighting.

**4.5.3 Other quantization schemes and network precisions.** BFAVerifier can be adapted to support other quantization schemes. For instance, when addressing a mixed-precision quantization scheme, only the weight interval associated with each symbolic parameter under bit-flip attacks needs to be adjusted for the BFA\_RA procedure, while no modifications are required for BFA\_MILP.

For floating-point neural networks (FPNNs), parameters are typically stored as IEEE 754 32-bit single-precision floating-point numbers, where flipping the exponent bit can cause drastic value changes (e.g., altering 0.078125 to  $1.25 \times 2^{24}$ ). BFAVerifier can be adapted to FPNNs by adjusting the parameter interval derived from bit-flipping, but the performance remains uncertain and is left for

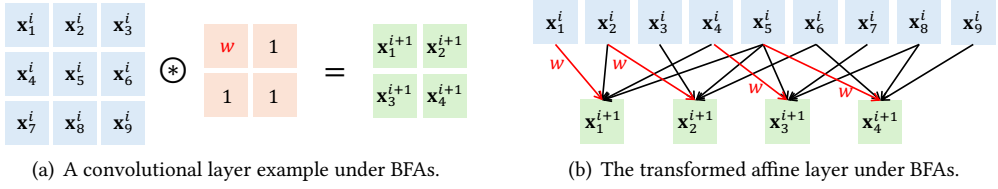


Fig. 6. An example of affine transformation from a convolutional computation under bit-flip attacks. Given a  $3 \times 3$  feature layer  $\mathbf{x}^i$  and a  $2 \times 2$  convolution filter with one parameter attacked (denoted as  $w$ ) and the other three parameters to be 1 in Figure 6(a), we can always get an equivalent affine layer as shown in Figure 6(b).

future work. Indeed, studies [16, 46] have shown that FPNNs are highly vulnerable to BFAs. Given this inherent vulnerability, we argue that verifying BFAs on QNNs is more reasonable, as FPNNs can be almost always compromised. Instead of formal verification, demonstrating vulnerabilities through attacks or testing is a more practical and insightful approach for FPNNs.

**Clarification.** Our method, BFAVerifier, is primarily designed to: i) effectively and efficiently prove desirable property of a given neural network under BFA and ii) or identify a relatively tight superset of vulnerable parameters that must be protected to avoid BFA. Note that the latter allows existing model integrity protection methods to be applied in a more cost-effectively way, i.e., by protecting only the identified vulnerable parameters (e.g., 0.01% of parameters in a network) rather than all parameters. Additionally, we consider the robustness of input regions against BFA, which is more interesting yet more challenging than the robustness of individual inputs. Note that it is virtually impossible to enumerate and verify all attack vectors for an input region and attackers may use BFAVerifier to identify critical bits/parameters to flip as well as an individual input.

## 5 Implementation and Evaluation

To validate the effectiveness of our method, we aim to answer the following research questions:

- RQ1.** How effective and efficient is BFA\_RA for providing the sound verification result on potential attack vectors, compared with the naive baseline method (cf. Section 3.2)?
- RQ2.** Can the absence of BFAs be verified with a conclusive result for a specific network using BFAVerifier, and how effective is the MILP-based method for providing a sound and complete verification result, as a complementary approach to BFA\_RA?
- RQ3.** How efficient and effective is BFAVerifier for verifying the absence of BFAs on larger networks with various activation functions?

**Implementation.** We implemented our verification method as an end-to-end tool BFAVerifier with Gurobi [26] as the back-end MILP solver. The SymPoly component is built upon GPUPoly [64], an open-source GPU implementation [52] of DeepPoly. The quantization follows the scheme mentioned in Section 2.1. During the network inference procedure and the verification process in BFAVerifier, quantized (fixed-point) parameters are stored in the IEEE 754 [1] floating-point number format for arithmetic operation. The floating-point number soundness flag in the implementation [52] of GPUPoly [64] is turned on for both DeepPoly and SymPoly.

**Datasets.** We use MNIST [41] and ACAS Xu [34] as the datasets in our experiments. MNIST contains 60,000 grayscale handwritten digits (from 0 to 9) of the size of  $28 \times 28$ . ACAS Xu is a safety-critical system designed to provide collision avoidance advisories for unmanned aircraft.

**Networks.** For the MNIST dataset, we train  $12 \times 2$  QNNs following the post-training quantization scheme [22, 44] on the MNIST [41] dataset, which is a common practice in prior research to improve the robustness against bit-flip attack [39, 68]. We evaluate 12 architectures with varying model

Table 2. QNNs obtained via quantization-aware training on MNIST with small network architectures.

ArchSmall	3blk_10	3blk_30	3blk_50	5blk_10	5blk_30	5blk_50	
# Param.	8.28k	26.62k	47.36k	8.5k	28.48k	52.46k	
Acc.	$Q = 4$	81.09%	94.40%	95.96%	86.80%	95.63%	96.44%
	$Q = 8$	92.83%	96.38%	96.38%	92.87%	96.95%	97.20%

Table 3. QNNs obtained via post-training quantization on MNIST with large network architectures.

ArchLarge	3blk_100	3blk_100 <sup>sigmoid</sup>	3blk_100 <sup>tan</sup>	3blk_512	3blk_1024	5blk_100	5blk_512	5blk_1024	
# Param.	109.7k	109.7k	109.7k	1,195k	3,962k	129.9k	1,720k	6,061k	
Acc.	$Q = 4$	97.03%	97.24%	97.43%	97.85%	97.72%	97.31%	97.62%	97.97%
	$Q = 8$	97.53%	97.71%	97.56%	98.18%	97.84%	97.39%	97.74%	98.06%

sizes and 2 quantization bit-widths  $Q \in \{4, 8\}$ , using ReLU activations by default. The details of the QNNs are listed in Tables 2 and 3. The first row shows the architecture of each QNN, where  $x\text{blk}_y$  means that the network has  $x$  hidden layers with each hidden layer containing  $y$  neurons. Row 2 shows the number of parameters in these networks and Rows 3-4 give the accuracy of these networks under different quantization bit-width, i.e.,  $Q = 4$  and  $Q = 8$ . Moreover, we consider two additional networks of architecture 3blk\_100 with Sigmoid and Tanh activation functions.

For the ACAS Xu dataset, although the authors in [36] provide 45 neural networks trained on this dataset along with 10 safety properties. We find that few of these properties can be proved via DeepPoly on these networks. Hence, in this work, we adopt retrained ones instead of the original networks from [57] as our benchmark. These retrained 45 networks adopt the same architecture as [36], i.e., 6blk\_50, and maintain comparable accuracy to the original networks (86.6% on average). These networks output a score for five different actions: clear-of-conflict (COC), weak left (WL), weak right (WR), strong left (SL), and strong right (SR). Based on these, we built 45 QNNs following a post-training quantization scheme, setting the quantization bit-width as  $Q = 8$ .

**Experimental setup.** For the BFA-tolerant robustness verification problem defined in Section 3.1, we randomly selected 20 inputs from the MNIST dataset for each network. We considered 3 different attack radii,  $r \in \{0, 2, 4\}$ , for each input, resulting in 60 input regions for each network. Note that all these input regions are robust to the corresponding QNNs until the bit-flip attacks. For the ACAS Xu benchmark, we test all 45 QNNs on the 10 properties and select the successfully proved 55 network-property pairs as our benchmarks. The details are given in Table 10 in Appendix A. We set the maximum number of bit flips as  $n \in \{1, 2, 4\}$ . Unless otherwise noted, each BFA\_RA task is conducted on an NVIDIA Tesla V100 accelerator and each BFA\_MILP task is conducted with 30 threads on a computer equipped with AMD EPYC 7742 64-core processors. The time limit for each verification task is 1 hour by default, considering the large number of tasks (thousands) in the subsequent experiments.

### 5.1 The Effectiveness and Efficiency of BFA\_RA

To answer **RQ1**, for each network listed in Tables 2 and 3, we randomly select 100 weight parameters and up to 100 bias parameters per layer for manipulation by the attacker, considering the huge amount of parameters in these networks. For each verification task of network with architecture  $x\text{blk}_y$  and quantized by  $Q$  bits, there are  $K \cdot \sum_{i=1}^n \binom{Q}{i}$   $(1, n)$ -attack vectors, where  $K = 100(x + 1) +$

Table 4. Verification results of BFA\_RA, BFA\_RA w.o binary search, and the naive method. Each entry shows the proportion of parameters that are proved to be safe/unknown by two compared methods. For example, the entry in the top left corner indicates that when  $r = 0, n = 1$ , there are 98.55% parameters across all verification tasks on all QNNs that are both proved as safe by the naive method and the BFA\_RA method. The bottom left corner entry indicates 0.03% of parameters are proven as unknown by the naive method but proved as safe by BFA\_RA, considering 8-bit quantization and robustness radius 4.

$Q = 4$		BFA_RA						BFA_RA w.o. Binary Search					
		#Safe_Paras			#Unknown_Paras			#Safe_Paras			#Unknown_Paras		
		$n = 1$	$n = 2$	$n = 4$	$n = 1$	$n = 2$	$n = 4$	$n = 1$	$n = 2$	$n = 4$	$n = 1$	$n = 2$	$n = 4$
Naive Method #Safe_Paras	$r = 0$	98.55%	98.13%	98.07%	0.00%	0.00%	0.00%	98.51%	98.00%	97.89%	0.04%	0.14%	0.17%
	$r = 2$	97.98%	97.48%	97.42%	0.00%	0.00%	0.00%	97.92%	97.33%	97.22%	0.06%	0.16%	0.20%
	$r = 4$	95.94%	95.19%	95.06%	0.00%	0.00%	0.00%	95.70%	94.71%	94.51%	0.24%	0.48%	0.55%
Naive Method #Unknown_Paras	$r = 0$	0.00%	0.00%	0.00%	1.45%	1.87%	1.93%	0.00%	0.00%	0.00%	1.45%	1.87%	1.93%
	$r = 2$	0.00%	0.01%	0.01%	2.02%	2.51%	2.57%	0.00%	0.00%	0.00%	2.02%	2.52%	2.58%
	$r = 4$	0.02%	0.06%	0.08%	4.05%	4.75%	4.85%	0.02%	0.04%	0.04%	4.05%	4.77%	4.89%

$Q = 8$		BFA_RA						BFA_RA w.o. Binary Search					
		#Safe_Paras			#Unknown_Paras			#Safe_Paras			#Unknown_Paras		
		$n = 1$	$n = 2$	$n = 4$	$n = 1$	$n = 2$	$n = 4$	$n = 1$	$n = 2$	$n = 4$	$n = 1$	$n = 2$	$n = 4$
Naive Method #Safe_Paras	$r = 0$	99.03%	98.59%	98.41%	0.00%	0.00%	0.00%	99.02%	98.54%	98.33%	0.01%	0.06%	0.08%
	$r = 2$	98.41%	97.89%	97.66%	0.00%	0.00%	0.00%	98.39%	97.83%	97.57%	0.02%	0.06%	0.09%
	$r = 4$	96.19%	95.43%	95.11%	0.00%	0.00%	0.00%	95.97%	94.95%	94.53%	0.21%	0.48%	0.59%
Naive Method #Unknown_Paras	$r = 0$	0.00%	0.00%	0.00%	0.97%	1.41%	1.59%	0.00%	0.00%	0.00%	0.97%	1.41%	1.59%
	$r = 2$	0.00%	0.00%	0.01%	1.59%	2.11%	2.33%	0.00%	0.00%	0.01%	1.59%	2.11%	2.34%
	$r = 4$	0.03%	0.08%	0.11%	3.78%	4.49%	4.78%	0.02%	0.05%	0.06%	3.79%	4.52%	4.82%

Table 5. The computation time (in GPU hours) of the three methods.

	Naive Method	BFA_RA	BFA_RA w.o. Binary Search
$Q = 4$	$\approx 72.9\text{h}$	$\approx 14.2\text{h}$	$\approx 13.6\text{h}$
$Q = 8$	$\approx 488.2\text{h}$	$\approx 14.8\text{h}$	$\approx 14.0\text{h}$

$x \min(y, 100) + 10$ . In total, we have  $28 \times 60 \times 3 = 5040$  verification tasks (28 networks, 60 input regions per network, and 3 different values of  $n$ ) for MNIST.

We compare the performance of BFA\_RA with the naive method mentioned in Section 3.2. Recall that, given a parameter, the naive method performs one reachability analysis for each  $(1, n)$ -attack vector to check robustness, whereas BFA\_RA performs one reachability analysis for all the possible  $(1, n)$ -attack vectors. Thus, BFA\_RA is expected to reduce execution time significantly. To evaluate the trade-off between efficiency and effectiveness in BFA\_RA compared to the naive method, we analyze the effectiveness loss/gain in Table 4 and the efficiency gain of BFA\_RA in Table 5, where

- Parameters proved as safe by the naive method but unknown by BFA\_RA indicate an **effectiveness loss** of BFA\_RA, i.e., row #Safe\_Paras and column #Unknown\_Paras.
- Parameters proved as safe by both methods demonstrate the **effectiveness maintenance** of BFA\_RA, i.e., row #Safe\_Paras and column #Safe\_Paras.
- Parameters proved as safe by BFA\_RA but unknown by the naive method represent an **effectiveness gain** of BFA\_RA, i.e., row #Unknown\_Paras and column #Safe\_Paras.

- Parameters proved as unknown by both methods reveals the **limitations** of reachability analysis by both methods, i.e., row #Unknown\_Paras and column #Unknown\_Paras.

Furthermore, to assess the effectiveness of the binary search strategy proposed in Section 4.3 for BFA\_RA, we implement a variant that excludes binary search, referred to as BFA\_RA w.o. Binary Search, where  $\text{binary\_RA}(\mathcal{N}, \mathcal{I}, g, w, w_l^+, w_u^+)$  (resp.  $\text{binary\_RA}(\mathcal{N}, \mathcal{I}, g, w, w_l^-, w_u^-)$ ) at line 3 (resp. line 5) in Algorithm 2 is replaced by  $\text{SymPoly}(\mathcal{N}, \mathcal{I}, g, w, w_l^+, w_u^+)$  (resp.  $\text{SymPoly}(\mathcal{N}, \mathcal{I}, g, w, w_l^-, w_u^-)$ ). The corresponding experimental results are also given in Tables 4 and 5.

*5.1.1 Effectiveness of BFA\_RA.* By analyzing the experimental results reported in Table 4, we observe that BFA\_RA consistently achieve an effectiveness gain over the naive method across all settings, including various radii of input perturbation  $r$  and maximal numbers of bit to flip  $n$ . Specifically, in cases where the naive method reports parameters as unknown (#Unknown\_Paras), BFA\_RA successfully verifies additional parameters as safe (#Safe\_Paras), albeit with a relatively modest gain (up to 0.11% on average). Indeed, it is reasonable that BFA\_RA achieves a relatively small effectiveness gain. In the worst case, BFA\_RA performs a binary search over each potential flipped weight value, similar to exhaustive traversal, making it at least as effective as the naive method. However, unlike the naive method, BFA\_RA initially treats each symbolic weight as a range and then partitions it using a binary approach. This introduced weight range may affect the reachability analysis of non-input neurons, leading to different abstract elements, i.e., distinct value domains obtained for each neuron between the naive method and SymPoly. In this setting, if a counterexample (a neuron value leading to a successful BFA) falls within the abstract element domain obtained by DeepPoly in the naive method but not within that of SymPoly, then BFA\_RA may exhibit an effectiveness gain. However, we argue that such cases should be rare, leading to a limited overall effectiveness gain of BFA\_RA over the naive method.

Furthermore, we also find that the binary-search-free variant of BFA\_RA demonstrates a reduction in effectiveness compared to BFA\_RA, as certain parameters that are proved as safe by both the naive method and BFA\_RA are proved as unknown by the variant. For example, when  $Q = 8$ ,  $r = 4$ , and  $n = 4$ , a total of 0.59% of the parameters that are verified as safe by both the naive method and BFA\_RA are proved as unknown by the binary-search-free variant of BFA\_RA.

**Result 1:** BFA\_RA demonstrates an effectiveness gain over both the naive method and its binary-search-free variant, albeit with a relatively modest improvement.

By comparing the experimental results between the naive method and binary-search-free variant of BFA\_RA, we find that only a small proportion of parameters (up to 0.55% for  $Q = 4$  and 0.59% for  $Q = 8$ ) are verified as safe by the naive method but remain unknown by the binary-search-free variant. Recall that the binary-search-free variant performs reachability analysis based on two intervals  $[w_l^+, w_u^+]$  and  $[w_l^-, w_u^-]$  (cf. line 2 in Algorithm 2), to approximate the reachability analysis result under bit-flip attack. The observed comparison results indicate that the abstract domain proposed in SymPoly effectively captures the impact of bit-flip operations on the corresponding parameters with high accuracy.

**Result 2:** The abstract domain proposed in SymPoly is relatively accurate in approximating the bit-flip operations.

*5.1.2 Efficiency of BFA\_RA.* By comparing the computation time of BFA\_RA and the naive method, as given in Table 5, we find that BFA\_RA consumes significantly less time than the naive method (up to 30x faster). It is noteworthy that the execution time for each query of DeepPoly and SymPoly is nearly identical. Therefore, the number of queries serves as a critical determinant of the overall efficiency of various methods. To illustrate it, we show the total number of queries invoked by

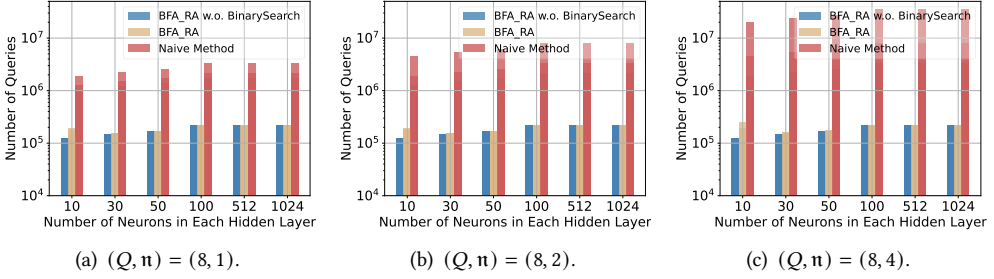


Fig. 7. The total number of queries with respect to DeepPoly or SymPoly in the three methods when  $Q = 8$ .

Table 6. Verification results of BFAVerifier on ACAS Xu

Property	BFA_RA		BFA_MILP		AvgTime(s)		#TO
	#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP	
Prop_3_WL	99.8%	0	0	24	356.0	4.2	0
Prop_3_WR	99.9%	0	1	26	354.7	4.0	0
Prop_3_SL	99.9%	0	0	23	354.8	44.2	1
Prop_3_SR	99.7%	0	0	27	357.5	5.9	0
Prop_5_SR	98.0%	0	0	20	381.2	45.3	1
Prop_10_COC	99.5%	11	2	13	365.2	227.7	16

the two methods in Figure 7 (we take  $Q = 8$  for example) and it can be observed that the naive method invokes an enormous amount of queries of DeepPoly, attributable to the fact that there are up to  $K \cdot \sum_{i=1}^n \binom{Q}{i}$  queries for each verification task. On the other hand, although the binary search strategy slightly increases the number of SymPoly queries, as shown in Figure 7, the execution time of BFA\_RA remains comparable to that of its binary-search-free variant (cf. Table 5).

**Result 3:** BFA\_RA is significantly more efficient than the naive method, achieving up to a 30x speedup. Moreover, it demonstrates comparable efficiency to its binary-search-free variant.

## 5.2 Verifying BFAs with BFAVerifier

To answer **RQ2**, in this section, we use BFAVerifier to verify the BFA-tolerant robustness property across all small networks outlined in Table 2 and all properties listed in Table 10. This results in a total of  $12 \times 60 \times 3 = 2160$  (12 networks, 60 input regions per network, and 3 different values of  $n$ ) verification tasks for the MNIST dataset and  $55 \times 3 = 165$  (55 network-property pairs and 3 different values of  $n$ ) verification tasks for the ACAS Xu dataset. It is important to note that, for each verification task, we assume that all model parameters, including weights and biases, are vulnerable to bit-flip attacks. Furthermore, we consider each attack to affect only a single parameter at a time, with the attacker potentially altering up to  $n \in \{1, 2, 4\}$  bits per attack. In the following, we define a task as successfully proved by BFA\_RA when all parameters in the corresponding network are proved as safe by BFA\_RA. Additionally, we consider a task as successfully solved by BFAVerifier when it is either proved by BFA\_RA or proved/falsified by BFA\_MILP.

**5.2.1 ACAS Xu.** The results for ACAS Xu are shown in Table 6 and Figure 8. In Table 6, Column 1 shows the property verified. Column 2 shows the average proportion of parameters that are proved to be safe by BFA\_RA across all corresponding networks and three different values of  $n$ . Column 3 gives the number of verification tasks that can be successfully verified by BFA\_RA. Columns 4 and



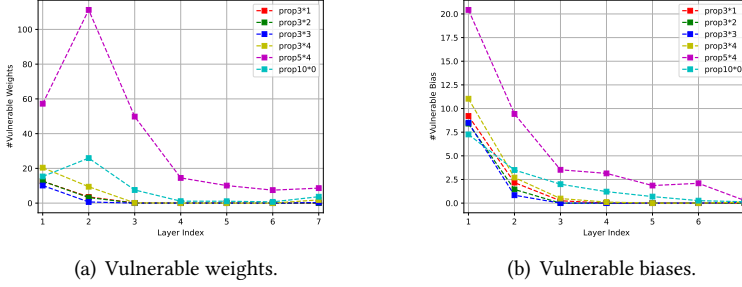


Fig. 8. The number of vulnerable parameters detected by BFA\_RA in each layer on ACAS Xu.

Table 7. Verification results of BFAVerifier on MNIST for small networks when  $(Q, r, n) = (8, 0, 1)$ .

Network	BFA_RA		BFA_MILP		AvgTime(s)		#TO
	#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP	
3blk_10	99.7%	0	0	20	30.4	0.2	0
3blk_30	99.9%	11	0	9	103.6	0.4	0
3blk_50	99.9%	19	1	0	203.6	0.7	0
5blk_10	99.2%	0	0	20	47.2	0.4	0
5blk_30	99.9%	17	3	0	171.9	0.4	0
5blk_50	99.9%	0	0	20	352.2	0.8	0

5 display the verification results by BFA\_MILP. Columns 6 and 7 give the average computation time for the two methods, and the last column gives the number of verification tasks that run out of time within 1 hour. We can observe that for ACAS Xu, BFAVerifier successfully solved 147 tasks, with 18 tasks running out of time within 1 hour. Among all of these, BFA\_RA proves 11 tasks independently. Note that when BFA\_RA w.o. binary search is used instead of BFA\_RA, the total number of proved tasks via pure reachability analysis decreases by 2, and the total number of timeout tasks (by BFA\_MILP) increases by 7. It is because that binary search strategy enables BFA\_RA to consistently obtain a tighter value range for each vulnerable parameter, leading to a more compact MILP model (or a reduced solution space) for the BFA\_MILP procedure and improving the overall efficiency. Detailed experimental results are presented in Table 12 in Appendix D.

Figures 8(a) and 8(b) show the detailed distribution of vulnerable weights and biases across all 7 non-input layers within the ACAS Xu networks, respectively. On average, we find that the proportion of vulnerable parameters in the earlier layers of the ACAS Xu networks is higher than that in the later layers. This observation suggests that enhanced protection measures should be prioritized for the parameters in the preceding layers to effectively mitigate the impact of bit-flip attacks. One possible reason behind this phenomenon is that the earlier layers of the ACAS Xu networks play a crucial role in feature extraction, making their parameters more susceptible to perturbations caused by BFAs. Since these layers directly influence the representations propagated through the network, any disruption in their parameters can have a cascading effect on the overall network performance, thereby increasing their vulnerability.

**5.2.2 MNIST.** For the MNIST benchmark, BFAVerifier successfully verifies the absence of BFAs for all 2160 tasks across all 12 QNNs. Due to space limitations, rather than presenting the average results across all possible values of  $Q$ ,  $r$ , and  $n$ , this section provides detailed verification results for a specific configuration of  $(Q, r, n) = (8, 0, 1)$  as an illustrative example.

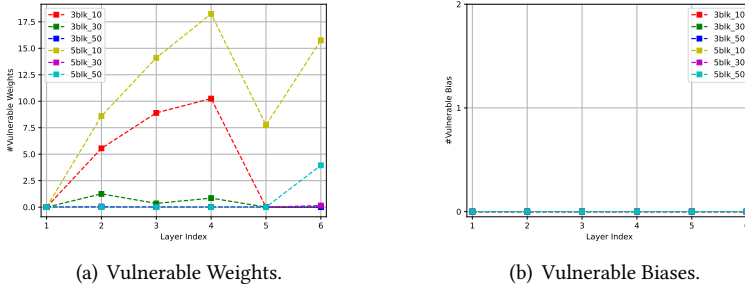


Fig. 9. The distribution of vulnerable parameters in each layer on MNIST when  $(Q, r, \mathfrak{n}) = (8, 0, 1)$ .

The results are presented in Table 13 and Figure 9. In Table 13, the first column lists the network architecture, while the remaining columns display the same types of results as those shown in Table 6. From Table 13, we can observe that for small networks, BFA\_RA achieves relatively high precision, as the majority of tasks (69 out of 73) that fail to be proved by BFA\_RA are indeed not robust to bit-flip attacks and subsequently falsified by BFA\_MILP.

Figures 9(a) and 9(b) give the average number of vulnerable weights and biases in each layer within the 8 networks, respectively. We observe a distinct phenomenon compared to ACAS Xu: in these small networks, the parameters in the middle layers exhibit greater vulnerability to bit-flip attacks. This suggests that greater attention should be given to protecting the middle layer to mitigate the impact of BFAs effectively. One possible explanation is that given the high input dimensions in these small networks, middle layers play a crucial role in transforming extracted features into high-level representations with fewer redundancy mechanisms to compensate for errors, making them more susceptible to BFAs. Furthermore, we find that the bias parameters of these small QNNs for MNIST exhibit significantly greater robustness against BFAs compared to those of QNNs for ACAS Xu.

**Result 4:** BFAVerifier can verify the absence of the bit-flip attacks, either prove the BFA-freeness or return a counter-example, and BFA\_MILP is effective as a complementary method to BFA\_RA.

### 5.3 BFAVerifier on Larger Networks with Various Activation Functions

To answer RQ3, in this subsection, we evaluate the performance of BFAVerifier on the larger networks listed in Table 3 resulting in a total of  $16 \times 60 \times 3 = 2880$  (16 networks, 60 input regions per network, and 3 different values of  $\mathfrak{n}$ ) verification tasks. Note that, although BFA\_RA is more efficient (in polynomial time to the network size) compared to the MILP-based method (which is NP-hard), it is still possible for the MILP-based method to effectively and efficiently verify BFA-tolerant robustness properties when the size of the input region and the vulnerable parameter set, i.e.,  $|\xi|$  in Algorithm 1, are limited.

We observe that, with the exceptions of networks under the 3blk\_100 and 5blk\_100 architectures, all other large networks listed in Table 3 cannot be successfully verified by BFAVerifier within 1 hour. For instance, a verification task for the network of architecture 3blk\_512\_512\_512 with  $(Q, r, \mathfrak{n}) = (8, 2, 1)$  requires approximately 14 hours to complete. Such failure is largely attributable to the challenges posed by i) the vast number of potential attack vectors and ii) the substantial model sizes when utilizing the MILP-based method, mirroring the issues found in existing MILP-based verification techniques in the context of vanilla network verification problems [32, 35].

Table 8. Verification results of BFAVerifier on the MNIST dataset for large networks, where each network undergoes 360 verification tasks in total.

	BFA_RA	BFA_MILP		#TO	#Solved
	#Proved	#Proved	#Falsified		
3blk_100	302	14	10	34	360
3blk_100 <sup>sigmoid</sup>	347	N/A	N/A	N/A	347
3blk_100 <sup>tanh</sup>	311	N/A	N/A	N/A	311
5blk_100	290	6	10	54	306

Table 9. Detailed verification results of BFAVerifier on 3blk\_100 and 5blk\_100 with  $Q \in \{4, 8\}$ .

	r	n	BFA_RA		BFA_MILP		AvgTime(s)		#TO	
			#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP		
$Q = 4$	0	1	100.0%	40	0	0	731.9	0	0	
		2	100.0%	40	0	0	749.8	0	0	
		4	100.0%	40	0	0	761.9	0	0	
	2	1	100.0%	40	0	0	2031.5	0	0	
		2	99.9%	38	1	1	2035.3	6.5	0	
		4	99.9%	38	1	1	2035.7	6.8	0	
	4	1	99.8%	29	3	0	2035.4	685.4	8	
		2	99.8%	19	3	3	2064.1	449.6	15	
		4	99.8%	19	3	2	2038.3	783.0	16	
	$Q = 8$	0	1	100.0%	40	0	0	722.2	0	0
			2	100.0%	40	0	0	745.0	0	0
			4	99.9%	39	0	1	754.2	1.6	0
2		1	99.9%	38	1	1	2032.5	2.8	0	
		2	99.9%	37	1	2	2038.6	6.9	0	
		4	99.9%	37	0	3	2046.4	2.6	0	
4		1	99.6%	22	3	1	2048.4	854.0	14	
		2	99.5%	18	3	2	2091.6	1482.3	17	
		4	99.4%	18	1	3	2173.7	297.7	18	

We report the experimental results in Tables 8 and 9. We find that BFAVerifier successfully solves the majority of verification tasks for the four networks in Table 8. For networks with logistic-based activations (3blk\_100<sup>sig</sup> and 3blk\_100<sup>tanh</sup>), although BFAVerifier can only provide sound verification results by exclusively utilizing BFA\_RA, BFAVerifier is still able to solve most tasks within the given time limit. We also find that, compared to small networks, larger networks appear to exhibit greater robustness against BFAs. Specifically, when  $r = 0$  (cf. Table 9), almost all BFA-tolerant properties can be successfully verified by BFA\_RA solely (except for one property when  $(Q, r, n) = (8, 0, 4)$ ), indicating enhanced resistance to bit-flip attacks within larger network architectures. Moreover, we observe that networks quantized with a lower bit-width tend to exhibit greater robustness against BFAs. This suggests that reduced bit-width quantization may inherently increase the difficulty of executing bit-flip attacks, a finding that aligns with the existing work [28].

**Result 5:** BFAVerifier demonstrates generalizability across various activation functions and scales to 8-bit QNNs with a 5blk\_100 architecture, completing verification of BFAs involving up to four bit flips within 1 hour.

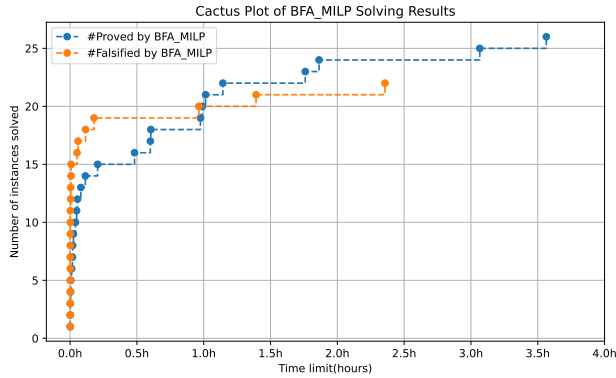


Fig. 10. The impact of time limit on the number of solved tasks by BFA\_MILP for experiments in Table 9.

For the unsolved tasks in Table 9, we further investigate how the number of verified tasks evolves with extended time limits (up to 4 hours). The results are illustrated in Figure 10. We observe that as the time limit increases, both the number of proved and the number of falsified tasks by BFA\_MILP increase, with the number of proved tasks exhibiting a more significant growth.

## 6 Related Work

In this section, we discuss the existing works closely related to the contributions of this paper.

**Verification of QNNs.** In the literature, quantization is broadly categorized into two types [22]: parameter-only quantization and quantization applied to both parameters and activations, leading to significant differences in verification methodologies. For parameter-only quantization, existing white-box DNN verification methods [25, 36, 45, 56, 75] can be applied directly, while primarily leverage constraint-solving or abstraction. A constraint-solving-based method reduces the verification problem into SMT/MILP solving [13, 20, 36, 37, 58]. While sound and complete, they often suffer from scalability limitations. To improve efficiency, various abstraction-based methods are proposed, such as computing a conservative bound of the output range based on different abstract domains [21, 45, 65, 66] or obtaining abstract neural networks, rendering them more suitable for verification [2, 19, 47, 55, 90]. A key distinction between SymPoly and other symbolic or polyhedral abstraction-based approaches in neural network verification is that existing methods abstract only neuron value ranges with fixed parameters, focusing solely on input interval propagation. In contrast, SymPoly extends abstraction to both neuron and parameter value ranges, enabling simultaneous propagation of both input and weight intervals. For QNNs where both parameters and activations are quantized, existing techniques primarily rely on constraint solving [23, 29, 31, 85] or BDD [86, 87], mainly for robustness properties.

**Bit-flip attacks and defense of neural networks.** DNNs are notably vulnerable to BFAs, where a single bit alternation can cause severe performance degradation [30, 46, 59]. To mitigate this, QNNs have been explored as a more resilient alternative. Building on the foundational work by Rakin et al. [61], a variety of attack technologies specifically designed for QNNs have then been investigated [7, 42, 49, 69]. These attacks primarily manipulate bits in non-volatile memory, affecting mainly the weights and occasionally the biases (e.g., DRAM), as well as introducing faults into certain neuron activation functions. Common BFA methodologies on networks include the Rowhammer attack [54, 61, 69], clock glitching attack [7], Voltage Frequency Scaling (VFS) attack [62], and lase

injection attack [18]. Notably, the Rowhammer, VFS, and laser inject attacks primarily manipulate the binary representations of weights and biases stored in memory [16, 42, 61, 62, 69], while the clock glitching attack specifically disrupts the functionality of the activation functions [7, 49].

The primary objective of defensive techniques is to enhance accuracy and/or robustness in the presence of BFAs. A natural approach to achieving this is to implement countermeasures against the underlying mechanisms that cause attacks, specifically by addressing the attacks from a hardware or system architecture perspective. For example, [80] selectively throttles memory accesses that could otherwise potentially cause Rowhammer bit-flips. Error correction Code [14, 15] is also an effective defense mechanism, typically implemented by the memory controller. However, no existing defense method can provide a definitive guarantee of eliminating all potential risks posed by bit-flip attacks.

**Other related works.** [76] proposes a method that formulates certified weight perturbations as an optimization problem, employing a uniform  $L_\infty$  norm perturbation within each layer. Their approach focuses on precision at the level of individual inputs, in contrast to ours BFAVerifier, which examines robustness against the BFAs over an input region. Another closely related study [77] investigates probabilistic safety verification of Bayesian networks utilizing weight interval propagation to identify disjoint safe weight spaces based on weight distributions. Although a direct comparison between their work and ours is not feasible due to the differences in network types and verification tasks, their methodology aligns with our naive abstraction approach depicted in Figure 3(a), which, as analyzed, exhibits lower abstract precision compared to the abstraction technique proposed in this work (cf. Figure 3(b)).

## 7 Conclusion and Future Work

We proposed a novel bit-flip attack verification method, BFAVerifier, for QNNs, which is sound, complete, and arguably efficient. To achieve this, we introduced SymPoly, the first abstract domain tailored for networks with symbolic parameters. We implemented BFAVerifier as an end-to-end tool and conducted thorough experiments on various benchmarks with networks of different model sizes and quantization bit-widths, demonstrating its effectiveness and efficiency.

While SymPoly may not represent the theoretically optimal abstract transformer for convex relaxations of weighted activation functions, it achieves optimal when restricting static abstractions to using only two linear constraints per neuron. Moreover, in terms of convex relaxation, the optimal abstraction transformer may not significantly enhance SymPoly, supported by the observation in [63] that there is an inherent *barrier* to tight relaxation-based verification methods. However, BFAVerifier could be integrated with complementary verification techniques, such as Branch-and-Bound for ReLU splitting [9] and optimizable lower bounds [79], to improve the verification precision and scalability. We consider these promising extensions for future work.

Our verification approach targets bit-flip attacks on a single parameter. Extending such threat models to attack multiple parameters simultaneously, e.g., two out of  $n$  parameters (i.e.,  $m = 2$ ), would involve a straightforward modification of Algorithm 1 by traversing all  $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$  two-parameter combinations in the for-loop at line 3, leading to exponential computation growth. Hence, although the abstract domain SymPoly can handle multiple symbolic parameters simultaneously, how to efficiently and effectively partition all  $\binom{n}{2}$  combinations into groups for abstraction-refinement poses a significant and non-trivial challenge, which is also a key focus in future work.

## 8 Data-Availability Statement

The source code of our tool and benchmarks are available at [4].

## Acknowledgments

This study was supported by the Strategic Priority Research Program of CAS (Award ID: XDA0320101), the Ministry of Education, Singapore under its Academic Research Fund Tier 2 (T2EP20222-0037), and the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (MOET32020-0003). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

## References

- [1] 2019. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), 1–84. <https://doi.org/10.1109/IEEESTD.2019.8766229>
- [2] Pranav Ashok, Vahid Hashemi, Jan Kretínský, and Stefanie Mohr. 2020. DeepAbstract: Neural Network Abstraction for Accelerating Verification. In *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis*. 92–107.
- [3] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. 2012. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proc. IEEE* 100, 11 (2012), 3056–3076.
- [4] BFAVerifier. 2025. <https://github.com/zhangyedi/BFAVerifier>.
- [5] Eli Biham and Adi Shamir. 1997. Differential Fault Analysis of Secret Key Cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference*. 513–525.
- [6] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 1997. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *Proceeding of the International Conference on the Theory and Application of Cryptographic Techniques*. 37–51.
- [7] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. 2018. Practical Fault Attack on Deep Neural Networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2204–2206. <https://doi.org/10.1145/3243734.3278519>
- [8] Lei Bu, Zhe Zhao, Yuchao Duan, and Fu Song. 2022. Taking Care of the Discretization Problem: A Comprehensive Study of the Discretization Problem and a Black-Box Adversarial Attack in Discrete Integer Domain. *IEEE Transactions on Dependable and Secure Computing* 19, 5 (2022), 3200–3217. <https://doi.org/10.1109/TDSC.2021.3088661>
- [9] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. 2018. A Unified View of Piecewise Linear Neural Network Verification. arXiv:1711.00455 [cs.AI] <https://arxiv.org/abs/1711.00455>
- [10] Guangke Chen, Sen Chen, Lingling Fan, Xiaoning Du, Zhe Zhao, Fu Song, and Yang Liu. 2021. Who is Real Bob? Adversarial Attacks on Speaker Recognition Systems. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy*.
- [11] Guangke Chen, Yedi Zhang, and Fu Song. 2024. SLMIA-SR: Speaker-Level Membership Inference Attacks against Speaker Recognition Systems. In *31st Annual Network and Distributed System Security Symposium*.
- [12] Guangke Chen, Yedi Zhang, Zhe Zhao, and Fu Song. 2023. QFA2SR: Query-Free Adversarial Transfer Attacks to Speaker Recognition Systems. In *32nd USENIX Security Symposium*, Joseph A. Calandrino and Carmela Troncoso (Eds.). 2437–2454.
- [13] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. 2017. Maximum Resilience of Artificial Neural Networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. 251–268.
- [14] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 55–71.
- [15] Andrea Di Dio, Koen Koning, Herbert Bos, and Cristiano Giuffrida. 2023. Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks.. In *NDSS*.
- [16] J. Dong, H. Qiu, Y. Li, T. Zhang, Y. Li, Z. Lai, C. Zhang, and S. Xia. 2023. One-bit Flip is All You Need: When Bit-flip Attack Meets Model Training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 4665–4675.
- [17] Shi Dong, Ping Wang, and Khushnood Abbas. 2021. A survey on deep learning and its applications. *Computer Science Review* 40 (2021), 100379.
- [18] Mathieu Dumont, Pierre-Alain Moëllic, Raphael Viera, Jean-Max Dutertre, and Rémi Bernhard. 2021. An Overview of Laser Injection against Embedded Neural Network Models. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. 616–621. <https://doi.org/10.1109/WF-IoT51360.2021.9595075>
- [19] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. 2020. An Abstraction-Based Framework for Neural Network Verification. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. 43–65.



- [20] Matteo Fischetti and Jason Jo. 2018. Deep neural networks and mixed integer linear optimization. *Constraints* 23, 3 (2018), 296–309.
- [21] Timon Gehr, Matthew Mirman, Dana Drachler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018.  $AI^2$ : Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. 3–18.
- [22] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*. Chapman and Hall/CRC, 291–326.
- [23] Mirco Giacobbe, Thomas A. Henzinger, and Mathias Lechner. 2020. How Many Bits Does it Take to Quantize Your Neural Network?. In *Proceedings of the 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 79–97. [https://doi.org/10.1007/978-3-030-45237-7\\_5](https://doi.org/10.1007/978-3-030-45237-7_5)
- [24] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 4851–4860. <https://doi.org/10.1109/ICCV.2019.00495>
- [25] Xingwu Guo, Wenjie Wan, Zhaodi Zhang, Min Zhang, Fu Song, and Xuejun Wen. 2021. Eager Falsification for Accelerating Robustness Verification of Deep Neural Networks. In *Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering*. 345–356.
- [26] Gurobi. 2018. A most powerful mathematical optimization solver. <https://www.gurobi.com/>.
- [27] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *Proceedings of the 4th International Conference on Learning Representations*.
- [28] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. 2020. Defending and harnessing the bit-flip based adversarial weight attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14095–14103.
- [29] Thomas A. Henzinger, Mathias Lechner, and Dorde Zikelic. 2021. Scalable Verification of Quantized Neural Networks. In *Proceedings of the 35th AAI Conference on Artificial Intelligence (AAAI)*. 3787–3795. <https://doi.org/10.1609/AAAI.V35I5.16496>
- [30] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. 2019. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 497–514.
- [31] Pei Huang, Haoze Wu, Yuting Yang, Ieva Daukantas, Min Wu, Yedi Zhang, and Clark Barrett. 2023. Towards Efficient Verification of Quantized Neural Networks. *arXiv preprint arXiv:2312.12679* (2023).
- [32] Pei Huang, Haoze Wu, Yuting Yang, Ieva Daukantas, Min Wu, Yedi Zhang, and Clark Barrett. 2024. Towards Efficient Verification of Quantized Neural Networks. In *Proceedings of the AAI Conference on Artificial Intelligence*, Vol. 38. 21152–21160.
- [33] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2704–2713.
- [34] Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. 2019. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (2019), 598–608.
- [35] Guy Katz, Clark Barrett, David L. Dill, Kyle D. Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification*. 97–117.
- [36] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification*. 97–117.
- [37] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proceedings of the 31st International Conference on Computer Aided Verification*. 443–452.
- [38] Faiq Khalid, Muhammad Abdullah Hanif, and Muhammad Shafique. 2021. Exploiting vulnerabilities in deep neural networks: Adversarial and fault-injection attacks. *arXiv preprint arXiv:2105.03251* (2021).
- [39] Navid Khoshavi, Mohammad Maghsoudloo, Arman Roohi, Saman Sargolzaei, and Yu Bi. 2022. HARDeNN: Hardware-assisted attack-resilient deep neural network architectures. *Microprocessors and Microsystems* 95 (2022), 104710.
- [40] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.

- [41] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database.
- [42] Kyungmi Lee and Anantha P. Chandrakasan. 2022. SparseBFA: Attacking Sparse Deep Neural Networks with the Worst-Case Bit Flips on Coordinates. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4208–4212. <https://doi.org/10.1109/ICASSP43922.2022.9747337>
- [43] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. 2019. Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification. In *Proceedings of the 26th International Symposium on Static Analysis*. 296–319.
- [44] Qun Li, Yuan Meng, Chen Tang, Jiacheng Jiang, and Zhi Wang. 2024. Investigating the Impact of Quantization on Adversarial Robustness. *arXiv preprint arXiv:2404.05639* (2024).
- [45] Renjue Li, Jianlin Li, Cheng-Chao Huang, Pengfei Yang, Xiaowei Huang, Lijun Zhang, Bai Xue, and Holger Hermanns. 2020. PRODeep: a platform for robustness verification of deep neural networks. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1630–1634.
- [46] Shaofeng Li, Xinyu Wang, Minhui Xue, Haojin Zhu, Zhi Zhang, Yansong Gao, Wen Wu, and Xuemin Sherman Shen. 2024. Yes, One-Bit-Flip Matters! Universal DNN Model Inference Depletion with Runtime Code Fault Injection. (2024).
- [47] Jiaxiang Liu, Yunhan Xing, Xiaomu Shi, Fu Song, Zhiwu Xu, and Zhong Ming. 2024. Abstraction and Refinement: Towards Scalable and Exact Verification of Neural Networks. *ACM Trans. Softw. Eng. Methodol.* 33, 5 (2024), 129:1–129:35. <https://doi.org/10.1145/3644387>
- [48] Liang Liu, Yanan Guo, Yueqiang Cheng, Youtao Zhang, and Jun Yang. 2023. Generating Robust DNN With Resistance to Bit-Flip Based Adversarial Weight Attack. *IEEE Trans. Comput.* 72, 2 (2023), 401–413. <https://doi.org/10.1109/TC.2022.3211411>
- [49] Wenyue Liu, Chip-Hong Chang, Fan Zhang, and Xiaoxuan Lou. 2020. Imperceptible misclassification attack on deep learning accelerator by glitch injection. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference (Virtual Event, USA) (DAC '20)*. IEEE Press, Article 29, 6 pages.
- [50] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. 2017. Fault injection attack on deep neural network. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 131–138.
- [51] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward ReLU neural networks. *CoRR* abs/1706.07351 (2017).
- [52] muellch, Gleb Makarchuk, GgnDp, skcho, Gagandeep Singh, François Serre, Tobias Zimmermann, Anian Ruoss, Mark Müller, Shachar Itzhaky, Jingxuan He, Haoze(Andrew) Wu, Isac Andrei, jorgenavas, Jose Calderon, and Jianlin Li. 2023. *eth-sri/ELINA*. <https://github.com/eth-sri/ELINA>
- [53] Abubakar Ahmad Musa, Adamu Hussaini, Weixian Liao, Fan Liang, and Wei Yu. 2023. Deep Neural Networks for Spatial-Temporal Cyber-Physical Systems: A Survey. *Future Internet* 15, 6 (2023), 199. <https://doi.org/10.3390/FI15060199>
- [54] Onur Mutlu and Jeremie S. Kim. 2020. RowHammer: A Retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2020), 1555–1571. <https://doi.org/10.1109/TCAD.2019.2915318>
- [55] Matan Ostrovsky, Clark W. Barrett, and Guy Katz. 2022. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. *CoRR* abs/2201.01978 (2022).
- [56] Long H Pham and Jun Sun. 2022. Verifying Neural Networks Against Backdoor Attacks. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV)*. 171–192. [https://doi.org/10.1007/978-3-031-13185-1\\_9](https://doi.org/10.1007/978-3-031-13185-1_9)
- [57] Long H Pham and Jun Sun. 2024. Certified Continual Learning for Neural Network Regression. *arXiv preprint arXiv:2407.06697* (2024).
- [58] Pavithra Prabhakar and Zahra Rahimi Afzal. 2019. Abstraction based Output Range Analysis for Neural Networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 15762–15772.
- [59] Cheng Qian, Ming Zhang, Yuanping Nie, Shuaibing Lu, and Huayang Cao. 2023. A Survey of Bit-Flip Attacks on Deep Neural Network and Corresponding Defense Methods. *Electronics* 12, 4 (2023). <https://doi.org/10.3390/electronics12040853>
- [60] Adnan Siraj Rakin, Md Hafizul Islam Chowdhury, Fan Yao, and Deliang Fan. 2022. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 1157–1174.
- [61] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-Flip Attack: Crushing Neural Network With Progressive Bit Search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- [62] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. 2021. Deep-Dup: An Adversarial Weight Duplication Attack Framework to Crush Deep Neural Network in Multi-Tenant FPGA. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1919–1936. <https://www.usenix.org/conference/usenixsecurity21/presentation/rakin>
- [63] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems* 32 (2019).

- [64] François Serre, Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2021. Scaling Polyhedral Neural Network Verification on GPUs. In *Proc. Machine Learning and Systems (MLSys'21)*.
- [65] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and Effective Robustness Certification. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 10825–10836.
- [66] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages (POPL)* 3 (2019), 41:1–41:30. <https://doi.org/10.1145/3290354>
- [67] Fu Song, Yusi Lei, Sen Chen, Lingling Fan, and Yang Liu. 2021. Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers. *Int. J. Intell. Syst.* 36, 9 (2021), 5210–5240.
- [68] David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. 2023. Random and Adversarial Bit Error Robustness: Energy-Efficient and Secure DNN Accelerators. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 3 (2023), 3632–3647. <https://doi.org/10.1109/TPAMI.2022.3181972>
- [69] M. Tol, S. Islam, A. J. Adiletta, B. Sunar, and Z. Zhang. 2023. Don't Knock! Rowhammer at the Backdoor of DNN Models. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, Los Alamitos, CA, USA, 109–122. <https://doi.org/10.1109/DSN58367.2023.00023>
- [70] M Caner Tol, Saad Islam, Berk Sunar, and Ziming Zhang. 2022. Toward realistic backdoor injection attacks on dnns using rowhammer. *arXiv preprint arXiv:2110.07683* (2022).
- [71] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *Proceedings of the International Conference on Computer Aided Verification*. 18–42.
- [72] Hoang-Dung Tran, Diego Manzananas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analysis of Deep Neural Networks. In *Proceedings of the 3rd World Congress on Formal Methods*. 670–686.
- [73] Andrew J Walker, Sungkwon Lee, and Dafna Beery. 2021. On DRAM rowhammer and the physics of insecurity. *IEEE Transactions on Electron Devices* 68, 4 (2021), 1400–1410.
- [74] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium '18*. 1599–1614.
- [75] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021).
- [76] Tsui-Wei Weng, Pu Zhao, Sijia Liu, Pin-Yu Chen, Xue Lin, and Luca Daniel. 2020. Towards certificated model robustness against weight perturbations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6356–6363.
- [77] Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. 2020. Probabilistic safety for bayesian neural networks. In *Conference on uncertainty in artificial intelligence*. PMLR, 1198–1207.
- [78] WikiChip. Accessed April 30, 2022. FSD Chip - Tesla. [https://en.wikichip.org/wiki/tesla\\_\(car\\_company\)/fsd\\_chip](https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip).
- [79] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. *arXiv:2011.13824 [cs.AI]* <https://arxiv.org/abs/2011.13824>
- [80] A Giray Yağlıkçı, Minesh Patel, Jeremie S Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, et al. 2021. Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 345–358.
- [81] Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. 2021. Improving Neural Network Verification through Spurious Region Guided Refinement. In *Proceedings of 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Jan Friso Groote and Kim Guldstrand Larsen (Eds.). 389–408.
- [82] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. 2020. DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*. 1463–1480.
- [83] Jinjie Zhang, Yixuan Zhou, and Rayan Saab. 2023. Post-training quantization for neural networks with provable guarantees. *SIAM Journal on Mathematics of Data Science* 5, 2 (2023), 373–399.
- [84] Yedi Zhang, Guangke Chen, Fu Song, Jun Sun, and Jin Song Dong. 2024. Certified Quantization Strategy Synthesis for Neural Networks. In *Proceedings of the 26th International Symposium on Formal Methods (FM), Part I*, André Platzer, Kristin Yvonne Rozier, Matteo Pradella, and Matteo Rossi (Eds.), Vol. 14933. 343–362. [https://doi.org/10.1007/978-3-031-71162-6\\_18](https://doi.org/10.1007/978-3-031-71162-6_18)
- [85] Yedi Zhang, Fu Song, and Jun Sun. 2023. QEBVerif: Quantization error bound verification of neural networks. In *International Conference on Computer Aided Verification*. Springer, 413–437.

- [86] Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, and Taolue Chen. 2021. BDD4BNN: A BDD-Based Quantitative Analysis Framework for Binarized Neural Networks. In *Proceedings of the 33rd International Conference on Computer Aided Verification (CAV)*. 175–200. [https://doi.org/10.1007/978-3-030-81685-8\\_8](https://doi.org/10.1007/978-3-030-81685-8_8)
- [87] Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, and Taolue Chen. 2023. Precise Quantitative Analysis of Binarized Neural Networks: A BDD-based Approach. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–51. <https://doi.org/10.1145/3563212>
- [88] Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, Min Zhang, Taolue Chen, and Jun Sun. 2022. QVIP: An ILP-based Formal Verification Approach for Quantized Neural Networks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 82:1–82:13. <https://doi.org/10.1145/3551349.3556916>
- [89] Zhe Zhao, Guangke Chen, Tong Liu, Taishan Li, Fu Song, Jingyi Wang, and Jun Sun. 2024. Attack as Detection: Using Adversarial Attack Methods to Detect Abnormal Examples. *ACM Trans. Softw. Eng. Methodol.* 33, 3 (2024), 68:1–68:45. <https://doi.org/10.1145/3631977>
- [90] Zhe Zhao, Yedi Zhang, Guangke Chen, Fu Song, Taolue Chen, and Jiaxiang Liu. 2022. CLEVEREST: Accelerating CEGAR-based Neural Network Verification via Adversarial Attacks. In *Proceedings of the 29th International Symposium on Static Analysis*. 449–473. [https://doi.org/10.1007/978-3-031-22308-2\\_20](https://doi.org/10.1007/978-3-031-22308-2_20)
- [91] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2022. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights. In *International Conference on Learning Representations*.

## A Overview of the ACAS Xu Benchmark

Table 10 gives the details of the ACAS Xu Benchmark we used throughout Section 5. We evaluate all 45 QNNs, denoted as  $N_{i,j}$  with  $1 \leq i \leq 5, 1 \leq j \leq 9$ , on the 10 properties, then select the successfully proved ones, resulting in a total of  $8 + 9 + 8 + 9 + 7 + 14 = 55$  network-property pairs as our benchmarks.

Table 10. Benchmarks of properties and QNNs obtained via post-quantization training for ACAS Xu.

Property	Description	Network
Prop_3_WL	If the intruder is directly ahead and is moving towards the ownership, a “Weak Left” maneuver is advised.	(8): $N_{2,6}, N_{2,7}, N_{2,8}, N_{2,9}, N_{4,6}, N_{4,7}, N_{4,8}, N_{4,9}$
Prop_3_WR	If the intruder is directly ahead and is moving towards the ownership, a “Weak Right” maneuver is advised.	(9): $N_{1,6}, N_{3,6}, N_{3,7}, N_{3,8}, N_{3,9}, N_{5,6}, N_{5,7}, N_{5,8}, N_{5,9}$
Prop_3_SL	If the intruder is directly ahead and is moving towards the ownership, a “Strong Left” maneuver is advised.	(8): $N_{2,2}, N_{2,3}, N_{2,4}, N_{2,5}, N_{4,2}, N_{4,3}, N_{4,4}, N_{4,5}$
Prop_3_SR	If the intruder is directly ahead and is moving towards the ownership, a “Strong Right” maneuver is advised.	(9): $N_{3,1}, N_{3,3}, N_{3,4}, N_{3,5}, N_{5,1}, N_{5,2}, N_{5,3}, N_{5,4}, N_{5,5}$
Prop_5_SR	If the intruder is near and approaching from the left, a “Strong Right” maneuver is advised.	(7): $N_{3,1}, N_{3,2}, N_{3,3}, N_{5,2}, N_{5,3}, N_{5,4}, N_{5,5}$
Prop_10_COC	For a far away intruder, a “Clear of Conflict” maneuver is advised.	(14): $N_{1,3}, N_{1,4}, N_{1,5}, N_{1,6}, N_{3,2}, N_{3,6}, N_{3,7}, N_{4,1}, N_{4,2}, N_{4,4}, N_{4,5}, N_{5,1}, N_{5,4}, N_{5,6}$

## B Missing Proofs in Sections 3 and 4

### B.1 Proof of Theorem 3.2

PROOF. To show that the problem of checking whether  $\mathcal{N} \models_{m,n}^{\rho} \langle \phi, \psi \rangle$  holds is in NP, we can

- (1) **Step 1:** non-deterministically guess an input  $\mathbf{x} \in \mathbb{R}^n$  and an  $(k, n)$ -attack vector  $\rho = \{(\alpha_1, P_1), \dots, (\alpha_k, P_k)\}$  for  $k \leq m$ ;
- (2) **Step 2:** build a new neural network  $\mathcal{N}^{\rho}$  according to the  $(k, n)$ -fault attack vector  $\rho$ ;
- (3) **Step 3:** compute  $\mathcal{N}^{\rho}(\mathbf{x})$  by feeding the values of the input  $\mathbf{x}$  forward through the network;
- (4) **Step 4:** check if both  $\phi(\mathbf{x})$  and  $\psi(\mathcal{N}^{\rho}(\mathbf{x}))$  hold, and return satisfiable if both  $\phi(\mathbf{x})$  and  $\psi(\mathcal{N}^{\rho}(\mathbf{x}))$  hold.

Since Steps 2–4 can be done in polynomial time, we conclude the proof.

The NP-hardness is proved by reducing from the satisfiability problem of the vanilla neural network verification problem  $\mathcal{N} \models \langle \phi, \psi \rangle$  which is NP-complete [35]. Consider a vanilla neural network verification problem of checking whether  $\mathcal{N} \models \langle \phi, \psi \rangle$  holds. Suppose the inputs and outputs of the neural network are  $\mathbf{x}$  and  $\mathbf{y} = \mathcal{N}(\mathbf{x})$ , respectively. We construct a neural network  $\mathcal{N}'$  as follows:

- $\mathcal{N}'$  comprises  $n + 1$  copies of the network verification  $\mathcal{N}$  in parallel,
- all the copies share the same inputs  $\mathbf{x}$ ,
- the outputs of the  $i$ -th copy are renamed by  $\mathbf{y}_i$ ,
- the weights of the edges between two neurons in two different copies are 0, ensuring that the neurons in the  $i$ -th copy are not affected by the neurons in other copies.

Let  $\psi' = \bigvee_{i=1}^{n+1} \psi_i$ , where  $\psi_i$  is obtained from the property  $\psi$  by renaming the outputs  $\mathbf{y}$  with the outputs  $\mathbf{y}_i$ .

**Claim.** For any fixed constants  $m$  and  $n$ ,  $\mathcal{N}' \models_{m,n}^{\rho} \langle \phi', \psi' \rangle$  holds iff  $\mathcal{N} \models \langle \phi, \psi \rangle$  holds.

( $\Leftarrow$ ) Suppose the vanilla neural network verification  $\mathcal{N} \models \langle \phi, \psi \rangle$  holds, then for any inputs  $\mathbf{x} \in \mathbb{R}^n$  that satisfies the pre-condition  $\phi$ ,  $\mathbf{y} = \mathcal{N}(\mathbf{x})$  also satisfies the post-condition  $\psi$ . According to the construction of  $\mathcal{N}'$ , for any  $(k, n)$ -fault attack vector  $\rho$  with  $k \leq m$ , there exists a copy of  $\mathcal{N}$ , say the  $i$ -th copy of  $\mathcal{N}$ , such that the outputs  $\mathbf{y}_i$  are the same as the outputs  $\mathbf{y}$ . It implies that  $\mathcal{N}'(\mathbf{x})$  satisfies  $\psi_i$ , hence  $\psi'$ . Thus,  $\mathcal{N}' \models_{m,n}^{\rho} \langle \phi', \psi' \rangle$  holds.

( $\Rightarrow$ ) Suppose the vanilla neural network verification problem  $\mathcal{N} \models \langle \phi, \psi \rangle$  does not hold, then there exists a counterexample  $\mathbf{x} \in \mathbb{R}^n$  such that  $\mathbf{x}$  satisfies the pre-condition  $\phi$  but  $\mathbf{y} = \mathcal{N}(\mathbf{x})$  does not satisfy the post-condition  $\psi$ . According to the construction of  $\mathcal{N}'$ , the outputs  $\mathcal{N}'(\mathbf{x})$  of  $\mathcal{N}'$  under an  $(m, 0)$ -fault attack vector  $\rho$  (i.e., no parameters can be changed) are  $n + 1$  copies of  $\mathbf{y} = \mathcal{N}(\mathbf{x})$ . Thus,  $\mathcal{N}'(\mathbf{x})$  does not satisfy  $\psi' = \bigvee_{i=1}^{n+1} \psi_i$ , i.e.,  $\mathcal{N}' \models_{m,n}^{\rho} \langle \phi', \psi' \rangle$  does not hold.  $\square$

## B.2 Proof of Theorem 4.1

**PROOF.** We first prove the soundness of the weighted-ReLU abstract transformer. Let  $\mathbf{a}_{k,1}^i = \langle a_{k,1}^{i,\leq}, a_{k,1}^{i,\geq}, l_{k,1}^i, u_{k,1}^i \rangle$  be the abstract element of  $\mathbf{x}_{k,1}^i$  and  $\mathbf{a}_{k,2}^i = \langle a_{k,2}^{i,\leq}, a_{k,2}^{i,\geq}, l_{k,2}^i, u_{k,2}^i \rangle$  be the abstract element of  $\mathbf{x}_{k,2}^i = \text{ReLU}(\mathbf{x}_{k,1}^i)$ .  $\gamma(\mathbf{a}_{k,1}^i) = \{x \in \mathbb{R} \mid a_{k,1}^{i,\leq} \leq x \leq a_{k,1}^{i,\geq}\}$ . Given  $\vec{\mathbf{W}}_{j,k}^{i+1} \in [w_l, w_u]$ , we prove the soundness by considering the following 5 cases:

- If  $l_{k,1}^i \geq 0$ :  $\text{ReLU}(\mathbf{x}_{k,1}^i) = \mathbf{x}_{k,1}^i$  for  $\mathbf{x}_{k,1}^i \in \gamma(\mathbf{a}_{k,1}^i)$  and  $a_{k,2}^{i,\leq} = a_{k,2}^{i,\geq} = \mathbf{x}_{k,1}^i$ . Then, we have  $w_l \cdot \mathbf{x}_{k,1}^i \leq \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i) \leq w_u \cdot \mathbf{x}_{k,1}^i$  and therefore

$$\begin{aligned} \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\gamma(\mathbf{a}_{k,1}^i)) &\subseteq \{x \in \mathbb{R} \mid w_l \cdot x' \leq x \leq w_u \cdot x' \wedge a_{k,1}^{i,\leq} \leq x' \leq a_{k,1}^{i,\geq}\} \\ &= \{x \in \mathbb{R} \mid w_l \cdot a_{k,2}^{i,\leq} \leq x \leq w_u \cdot a_{k,2}^{i,\geq}\} = \gamma(\mathbf{a}_{k,2}^{i,*}) \end{aligned}$$

- If  $u_{k,1}^i \leq 0$ :  $\text{ReLU}(\mathbf{x}_{k,1}^i) = 0$  for  $\mathbf{x}_{k,1}^i \in \gamma(\mathbf{a}_{k,1}^i)$  and  $a_{k,2}^{i,\leq} = a_{k,2}^{i,\geq} = 0$ . Then, we have  $0 \leq \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i) \leq 0$  and therefore

$$\begin{aligned} \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\gamma(\mathbf{a}_{k,1}^i)) &\subseteq \{x \in \mathbb{R} \mid 0 \leq x \leq 0 \wedge a_{k,1}^{i,\leq} \leq x' \leq a_{k,1}^{i,\geq}\} \\ &= \{x \in \mathbb{R} \mid w_l \cdot a_{k,2}^{i,\leq} \leq x \leq w_u \cdot a_{k,2}^{i,\geq}\} = \gamma(\mathbf{a}_{k,2}^{i,*}) \end{aligned}$$

- If  $l_{k,1}^i u_{k,1}^i < 0$ : we have  $\lambda \cdot \mathbf{x}_{k,1}^i \leq \text{ReLU}(\mathbf{x}_{k,1}^i) \leq \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i)$  for  $\mathbf{x}_{k,1}^i \in \gamma(\mathbf{a}_{k,1}^i)$ .  $a_{k,2}^{i,\leq} = \lambda \cdot \mathbf{x}_{k,1}^i$  and  $a_{k,2}^{i,\geq} = \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i)$ .



– When  $w_l \geq 0$ : we have  $w_l \cdot \lambda \cdot \mathbf{x}_{k,1}^i \leq \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i) \leq w_u \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i)$  and therefore

$$\begin{aligned} \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\gamma(\mathbf{a}_{k,1}^i)) &\subseteq \{x \in \mathbb{R} \mid w_l \cdot \lambda \cdot x' \leq x \leq w_u \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (x' - l_{k,1}^i) \wedge a_{k,1}^{i,\leq} \leq x' \leq a_{k,1}^{i,\geq}\} \\ &= \{x \in \mathbb{R} \mid w_l \cdot a_{k,2}^{i,\leq} \leq x \leq w_u \cdot a_{k,2}^{i,\geq}\} \\ &= \{x \in \mathbb{R} \mid \tilde{a}_{k,2}^{i,\leq} \leq x \leq \tilde{a}_{k,2}^{i,\geq}\} = \gamma(\mathbf{a}_{k,2}^{i,*}) \end{aligned}$$

– When  $w_u \leq 0$ : we have  $w_l \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i) \leq \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i) \leq w_u \cdot \lambda \cdot \mathbf{x}_{k,1}^i$  and therefore

$$\begin{aligned} \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\gamma(\mathbf{a}_{k,1}^i)) &\subseteq \{x \in \mathbb{R} \mid w_l \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (x' - l_{k,1}^i) \leq x \leq w_u \cdot \lambda \cdot x' \wedge a_{k,1}^{i,\leq} \leq x' \leq a_{k,1}^{i,\geq}\} \\ &= \{x \in \mathbb{R} \mid w_l \cdot a_{k,2}^{i,\geq} \leq x \leq w_u \cdot a_{k,2}^{i,\leq}\} = \{x \in \mathbb{R} \mid \tilde{a}_{k,2}^{i,\leq} \leq x \leq \tilde{a}_{k,2}^{i,\geq}\} = \gamma(\mathbf{a}_{k,2}^{i,*}) \end{aligned}$$

– When  $w_l < 0 < w_u$ : if  $\lambda = 0$ , we have  $0 \leq \text{ReLU}(\mathbf{x}_{k,1}^i) \leq \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i)$ . Hence, we have

$$w_l \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i) \leq \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i) \leq w_u \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i). \text{ If } \lambda = 1, \text{ we have } \mathbf{x}_{k,1}^i \leq \text{ReLU}(\mathbf{x}_{k,1}^i) \leq \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i). \text{ Since } w_l < 0, \text{ we have } w_l \cdot \mathbf{x}_{k,1}^i \geq w_l \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i).$$

Finally, we have  $w_l \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i) \leq \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\mathbf{x}_{k,1}^i) \leq w_u \cdot \frac{u_{k,1}^i}{u_{k,1}^i - l_{k,1}^i} (\mathbf{x}_{k,1}^i - l_{k,1}^i)$  for  $\lambda \in \{0, 1\}$  and therefore

$$\begin{aligned} \vec{\mathbf{W}}_{j,k}^{i+1} \cdot \text{ReLU}(\gamma(\mathbf{a}_{k,1}^i)) &\subseteq \{x \in \mathbb{R} \mid w_l \cdot \frac{u_{k,1}^i (x' - l_{k,1}^i)}{u_{k,1}^i - l_{k,1}^i} \leq x \leq w_u \cdot \frac{u_{k,1}^i (x' - l_{k,1}^i)}{u_{k,1}^i - l_{k,1}^i} \wedge a_{k,1}^{i,\leq} \leq x' \leq a_{k,1}^{i,\geq}\} \\ &= \{x \in \mathbb{R} \mid w_l \cdot a_{k,2}^{i,\geq} \leq x \leq w_u \cdot a_{k,2}^{i,\leq}\} = \{x \in \mathbb{R} \mid \tilde{a}_{k,2}^{i,\leq} \leq x \leq \tilde{a}_{k,2}^{i,\geq}\} = \gamma(\mathbf{a}_{k,2}^{i,*}) \end{aligned}$$

Therefore, **our weighted-ReLU abstract transformer is sound** in all cases. We then prove that the invariant preserving. Let  $\gamma(\mathbf{a}_{k,2}^{i,*}) = \{x \in \mathbb{R} \mid \tilde{a}_{k,2}^{i,\leq} \leq x \leq \tilde{a}_{k,2}^{i,\geq}\}$  and  $l_{k,2}^i \leq a_{k,2}^{i,\leq} \leq \mathbf{x}_{k,2}^i \leq a_{k,2}^{i,\geq} \leq u_{k,2}^i$ :

- If  $w_l \geq 0$ , we have  $w_l \cdot l_{k,2}^i \leq w_l \cdot a_{k,2}^{i,\leq}$  and  $w_u \cdot a_{k,2}^{i,\geq} \leq w_u \cdot u_{k,2}^i$ . Then, we have  $\tilde{l}_{k,2}^i = w_l \cdot l_{k,2}^i \leq w_l \cdot a_{k,2}^{i,\leq} = \tilde{a}_{k,2}^{i,\leq} \leq \tilde{x}_{k,2}^i \leq \tilde{a}_{k,2}^{i,\geq} = w_u \cdot a_{k,2}^{i,\geq} \leq w_u \cdot u_{k,2}^i = \tilde{u}_{k,2}^i$ ;
- If  $w_u \leq 0$ , we have  $w_l \cdot u_{k,2}^i \leq w_l \cdot a_{k,2}^{i,\geq}$  and  $w_u \cdot a_{k,2}^{i,\leq} \leq w_u \cdot l_{k,2}^i$ . Then, we have  $\tilde{l}_{k,2}^i = w_l \cdot u_{k,2}^i \leq w_l \cdot a_{k,2}^{i,\geq} = \tilde{a}_{k,2}^{i,\leq} \leq \tilde{x}_{k,2}^i \leq \tilde{a}_{k,2}^{i,\geq} = w_u \cdot a_{k,2}^{i,\leq} \leq w_u \cdot l_{k,2}^i = \tilde{u}_{k,2}^i$ ;
- If  $w_l < 0 < w_u$ , we have  $w_l \cdot u_{k,2}^i \leq w_l \cdot a_{k,2}^{i,\geq}$  and  $w_u \cdot a_{k,2}^{i,\leq} \leq w_u \cdot u_{k,2}^i$ . Then, we have  $\tilde{l}_{k,2}^i = w_l \cdot u_{k,2}^i \leq w_l \cdot a_{k,2}^{i,\geq} = \tilde{a}_{k,2}^{i,\leq} \leq \tilde{x}_{k,2}^i \leq \tilde{a}_{k,2}^{i,\geq} = w_u \cdot a_{k,2}^{i,\leq} \leq w_u \cdot u_{k,2}^i = \tilde{u}_{k,2}^i$ ;

Therefore, **our weight-ReLU abstract transformer preserves the invariant** in all cases, and we finish the proving.  $\square$

### B.3 Proof of Theorem 4.2

PROOF. Consider  $\mathbf{x}_{j,1}^2 = \sum_{t \in [n_2] \setminus k} \mathbf{W}_{j,t}^2 \mathbf{x}_t^1 + \vec{\mathbf{W}}_{j,k}^2 \mathbf{x}_k^1 + \mathbf{b}_j^2$ , we can inherit the proof of affine abstract transformer in [66] directly on all input neurons  $\mathbf{x}_t^1$  for  $t \in [n_2] \setminus k$ . To prove the theorem, it remains to demonstrate that the abstract transformer for the weighted input neuron  $\vec{\mathbf{W}}_{j,k}^2 \mathbf{x}_k^1$  is sound, i.e., to prove that the abstraction  $\langle a^{\leq}, a^{\geq}, l, u \rangle$  preserves all possible concrete values of  $\vec{\mathbf{W}}_{j,k}^2 \mathbf{x}_k^1$



with the input range of  $\mathbf{x}_k^1$  as  $[x_l, x_u]$  and value range of  $\vec{\mathbf{W}}_{j,k}^2$  as  $[w_l, w_u]$ , where  $a^{\leq} = \kappa^{\leq} \mathbf{x}_k^1 - \eta$ ,  $a^{\geq} = \kappa^{\leq} \mathbf{x}_k^1 + \eta$ , and:

- If  $x_l \geq 0$ , then  $\{\kappa^{\leq} = w_l, \kappa^{\geq} = w_u, \eta = 0\}$ ;
- If  $x_u \leq 0$ , then  $\{\kappa^{\leq} = w_u, \kappa^{\geq} = w_l, \eta = 0\}$ ;
- Otherwise,  $\{\kappa^{\leq} = \frac{w_u x_u - w_l x_l}{x_u - x_l}, \kappa^{\geq} = \frac{w_l x_u - w_u x_l}{x_u - x_l}, \eta = \frac{x_u x_l}{x_u - x_l} (w_l - w_u)\}$ .

Intuitively, an intuitive sound abstract transformer for the function  $y = \vec{\mathbf{W}}_{j,k}^2 \mathbf{x}_k^1$  considering  $\mathbf{x}_k^1 \in [x_l, x_u]$  and  $\vec{\mathbf{W}}_{j,k}^2 \in [w_l, w_u]$  is the convex quadrilateral constructed by the four vertices:  $A = (x_l, w_l x_l)$ ,  $B = (x_u, w_l x_u)$ ,  $C = (x_l, w_u x_l)$ , and  $D = (x_u, w_u x_u)$ . The upper and lower boundaries of the quadrilateral are shown in Table 11.

Table 11. Boundaries of the quadrilateral constructed by the four vertices:  $A = (x_l, w_l x_l)$ ,  $B = (x_u, w_l x_u)$ ,  $C = (x_l, w_u x_l)$ , and  $D = (x_u, w_u x_u)$ , where  $\overline{AB}$  denote the boundary is the line segment of AB.

	$w_l \geq 0$			$w_u \leq 0$			$w_l < 0 < w_u$		
	$x_l \geq 0$	$x_u \leq 0$	$x_l < 0 < x_u$	$x_l \geq 0$	$x_u \leq 0$	$x_l < 0 < x_u$	$x_l \geq 0$	$x_u \leq 0$	$x_l < 0 < x_u$
Upper Boundary	$\overline{CD}$	$\overline{AB}$	$\overline{AD}$	$\overline{CD}$	$\overline{AB}$	$\overline{AD}$	$\overline{CD}$	$\overline{AB}$	$\overline{AD}$
Lower Boundary	$\overline{AB}$	$\overline{CD}$	$\overline{BC}$	$\overline{AB}$	$\overline{CD}$	$\overline{BC}$	$\overline{AB}$	$\overline{CD}$	$\overline{BC}$

For any  $x \in [x_l, x_u]$ , the segment  $\overline{AB}$  is given by  $y = w_l x$ ,  $\overline{CD}$  is given by  $y = w_u x$ ,  $\overline{AD}$  is given by  $y = w_l x_l + \frac{w_u x_u - w_l x_l}{x_u - x_l} (x - x_l)$ , and  $\overline{BC}$  is given by  $y = w_l x_u + \frac{w_l x_u - w_u x_l}{x_u - x_l} (x - x_u)$ . Since these expressions fully characterize the convex hull of the function over the given input interval, the theorem is thereby proved.  $\square$

#### B.4 Proof of Theorem 4.3

PROOF. Since the abstraction loses no precision, soundness and the invariant are preserved directly.  $\square$

#### B.5 Proof of Theorem 4.5

PROOF. We prove it by construction.

Assuming that the abstract element of  $g(x)$  obtained in DeepPoly is  $\langle a^{\leq}, a^{\geq}, l, u \rangle$ , let  $\langle \tilde{a}^{\leq}, \tilde{a}^{\geq}, \tilde{l}, \tilde{u} \rangle$  denote the abstract element of  $w \cdot g(x)$ . We use  $g'(x)$  and  $g''(x)$  to denote the first and second derivatives of  $g(x)$ . Next, we prove the soundness of the abstract transformer demonstrated in Table 1 by construction based on the existing proof ideas on the abstract transformers of the Sigmoid and Tanh activation functions in [66], which can be illustrated as follows:

- When  $l \geq 0$ ,  $a^{\leq}$  is given by the line segment defined by two points  $(l, g(l))$  and  $(u, g(u))$ , i.e., the slope is  $\kappa = \frac{g(u) - g(l)}{u - l}$ . This is because  $g$  is concave on  $[l, u]$ ; Otherwise,  $a^{\leq}$  is given by the function defined by the point  $(l, g(l))$  and a minimum slope  $\kappa' = \min(g'(l), g'(u))$ , i.e.,  $y(x) = \kappa'(x - l) + g(l)$ . This is because  $g'$  is non-decreasing on  $(l, 0]$  and decreasing on  $[0, u)$ , then by setting the slope as  $\kappa'$ , we can always guarantee that  $\kappa'$  is the minimum derivative for any point on  $g(x)$  with  $x \in [l, u]$ , hence for any point  $x_c \in [l, u]$ ,  $\kappa'(x - x_c) + g(x_c)$  always lies below of  $g(x)$  on  $[x_c, u]$ . Hence, when  $x_c = l$ , we have  $\kappa'(x - l) + g(l)$  always lies below of  $g(x)$  on  $[l, u]$ .
- When  $u \leq 0$ ,  $a^{\geq}$  is given by the line segment defined by two points  $(l, g(l))$  and  $(u, g(u))$ , i.e., the slope is  $\kappa = \frac{g(u) - g(l)}{u - l}$ . This is because  $g$  is convex on  $[l, u]$ ; Otherwise,  $a^{\geq}$  is given by the function defined by the point  $(u, g(u))$  and the minimum slope  $\kappa' = \min(g'(l), g'(u))$ , i.e.,

$y(x) = \kappa'(x - u) + g(u)$ . This is because  $g'$  is non-decreasing on  $(l, 0]$  and decreasing on  $[0, u]$ , then by setting the slope as  $\kappa'$ , we can always guarantee that  $\kappa'$  is the minimum derivative for any point on  $g(x)$  with  $x \in [l, u]$ , hence for any point  $x_c \in [l, u]$ ,  $\kappa'(x - x_c) + g(x_c)$  always lies above of  $g(x)$  on  $[l, x_c]$ . Hence, when  $x_c = u$ , we have  $\kappa'(x - u) + g(u)$  always lies above of  $g(x)$  on  $[l, u]$ .

**We first prove the theorem when  $g(x) = \text{Sigmoid}(x)$  by construction.**

Assuming that  $a_{w_u}^{\geq}$  is the upper boundary of  $w_u \cdot \text{Sigmoid}(x)$ , then we have  $a_{w_u}^{\geq}(x) \geq w_u \cdot \text{Sigmoid}(x) \geq w \cdot \text{Sigmoid}(x)$  with  $w \in [w_l, w_u] \wedge x \in [l, u]$ . Similarly, assuming  $a_{w_l}^{\leq}$  is the lower boundary of  $w_l \cdot \text{Sigmoid}(x)$ , then we have  $a_{w_l}^{\leq}(x) \leq w_l \cdot \text{Sigmoid}(x) \leq w \cdot \text{Sigmoid}(x)$  with  $w \in [w_l, w_u] \wedge x \in [l, u]$ . Hence,  $a_{w_l}^{\leq}$  and  $a_{w_u}^{\geq}$  are sound lower and upper boundaries, i.e.,  $\tilde{a}^{\leq}$  and  $\tilde{a}^{\geq}$ , of abstract domain  $w \cdot \text{Sigmoid}(x)$  with  $w \in [w_l, w_u]$  on  $x \in [l, u]$ , respectively. By following the above abstract element construction idea from [66], we obtain  $\tilde{a}^{\leq} = a_{w_l}^{\leq}$  and  $\tilde{a}^{\geq} = a_{w_u}^{\geq}$  as follows:

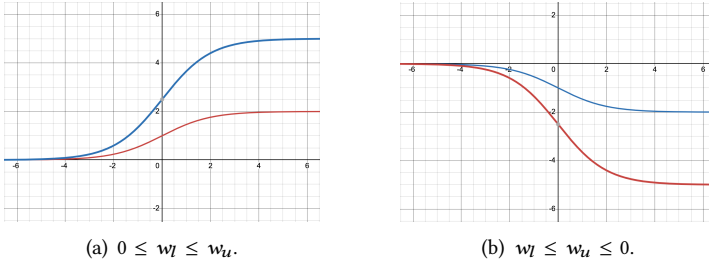


Fig. 11.  $w \cdot \text{Sigmoid}(x)$  with  $w \in \{w_l, w_u\}$ ,  $w_l$  in red and  $w_u$  in blue.

- When  $w_l \geq 0$  (cf. Figure 11(a)), we have  $\tilde{l} = w_l g(l)$ ,  $\tilde{u} = w_u g(u)$ , and
  - If  $l \geq 0$ ,  $a_{w_l}^{\leq}$  is given by the line segment defined by two points  $(l, w_l g(l))$  and  $(u, w_l g(u))$ , i.e.,  $a_{w_l}^{\leq} = w_l g(l) + \frac{w_l g(u) - w_l g(l)}{u - l} (x - l) = w_l g(l) + w_l \kappa (x - l)$ ; Otherwise,  $a_{w_l}^{\leq}$  is given by the function defined by the point  $(l, w_l g(l))$  and the minimum slope of  $w_l g(x)$  on  $[l, u]$ , i.e.,  $a_{w_l}^{\leq} = w_l g(l) + \min((w_l g(x))' |_{x=u}, (w_l g(x))' |_{x=l}) (x - l) = w_l g(l) + w_l \min(g'(u), g'(l)) (x - l) = w_l g(l) + w_l \kappa' (x - l)$ ;
  - If  $u \leq 0$ ,  $a_{w_u}^{\geq}$  is given by the line segment defined by two points  $(l, w_u g(l))$  and  $(u, w_u g(u))$ , i.e.,  $a_{w_u}^{\geq} = w_u g(u) + \frac{w_u g(l) - w_u g(u)}{l - u} (x - u) = w_u g(u) + w_u \kappa (x - u)$ ; Otherwise,  $a_{w_u}^{\geq}$  is given by the function defined by the point  $(u, w_u g(u))$  and the minimum slope of  $w_u g(x)$  on  $[l, u]$ , i.e.,  $a_{w_u}^{\geq} = w_u g(u) + \min((w_u g(x))' |_{x=u}, (w_u g(x))' |_{x=l}) (x - u) = w_u g(u) + w_u \min(g'(u), g'(l)) (x - u) = w_u g(u) + w_u \kappa' (x - u)$ ;
- When  $w_u \leq 0$  (cf. Figure 11(b)), we have  $\tilde{l} = w_l g(u)$ ,  $\tilde{u} = w_u g(l)$ , and
  - If  $u \leq 0$ ,  $a_{w_l}^{\leq}$  is given by the line segment defined by two points  $(l, w_l g(l))$  and  $(u, w_l g(u))$ , i.e.,  $a_{w_l}^{\leq} = w_l g(u) + \frac{w_l g(l) - w_l g(u)}{u - l} (x - u) = w_l g(u) + w_l \kappa (x - u)$ ; Otherwise,  $a_{w_l}^{\leq}$  is given by the function defined by the point  $(u, w_l g(u))$  and the maximum slope of  $w_l g(x)$  on  $[l, u]$ , i.e.,  $a_{w_l}^{\leq} = w_l g(u) + \max((w_l g(x))' |_{x=u}, (w_l g(x))' |_{x=l}) (x - u) = w_l g(u) + w_l \min(g'(u), g'(l)) (x - u) = w_l g(u) + w_l \kappa' (x - u)$ ;
  - If  $l \geq 0$ ,  $a_{w_u}^{\geq}$  is given by the line segment defined by two points  $(l, w_u g(l))$  and  $(u, w_u g(u))$ , i.e.,  $a_{w_u}^{\geq} = w_u g(l) + \frac{w_u g(u) - w_u g(l)}{u - l} (x - l) = w_u g(l) + w_u \kappa (x - l)$ ; Otherwise,  $a_{w_u}^{\geq}$  is given by the function defined by the point  $(l, w_u g(l))$  and the maximum slope of  $w_u g(x)$  on  $[l, u]$ , i.e.,

$$a_{w_u}^{\geq} = w_u g(l) + \max((w_u g(x))' |_{x=u}, (w_u g(x))' |_{x=l})(x-l) = w_u g(l) + w_u \min(g'(u), g'(l))(x-l) \\ = w_u g(l) + w_u \kappa'(x-l);$$

Finally, given the weighted Sigmoid function  $\vec{W}_{j,k}^{i+1} \cdot \text{Sigmoid}(x_{k,1}^i)$ , we can construct and obtain its sound abstract element presented in Table 1.

**We next prove the theorem when  $g(x) = \text{Tanh}(x)$  by construction.** Note that the sign of the value changes when crossing  $x = 0$  in the Tanh function. To ensure soundness, we construct the abstract element based on different values of  $x$  and  $w$  directly as follows:

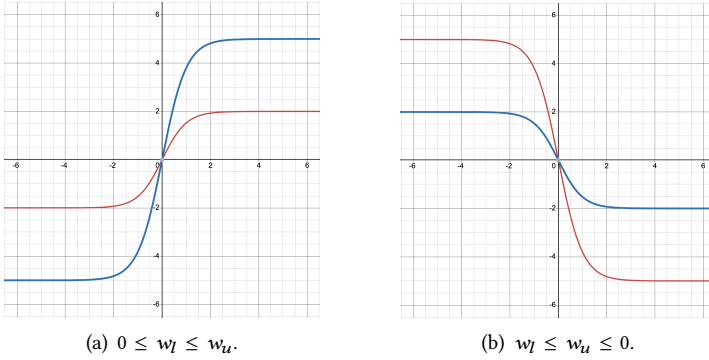


Fig. 12.  $w \cdot \text{Tanh}(x)$  with  $w \in \{w_l, w_u\}$ ,  $w_l$  in red and  $w_u$  in blue.

- When  $w_l \geq 0$  (cf. Figure 12(a)):
  - If  $l \geq 0$ , then  $\tilde{l} = w_l g(l)$ ,  $\tilde{u} = w_u g(u)$ ,  $\tilde{a}^{\leq} = a_{w_l}^{\leq}$ , and  $\tilde{a}^{\geq} = a_{w_u}^{\geq}$ , where  $a_{w_l}^{\leq} = w_l g(l) + w_l \kappa(x-l)$  and  $a_{w_u}^{\geq} = w_u g(u) + w_u \kappa'(x-u)$ ;
  - If  $u \leq 0$ , then  $\tilde{l} = w_u g(l)$ ,  $\tilde{u} = w_l g(u)$ ,  $\tilde{a}^{\leq} = a_{w_u}^{\leq}$ , and  $\tilde{a}^{\geq} = a_{w_l}^{\geq}$ , where  $a_{w_u}^{\leq} = w_u g(l) + w_u \kappa'(x-l)$  and  $a_{w_l}^{\geq} = w_l g(u) + w_l \kappa(x-u)$ ;
  - If  $l < 0 < u$ , then  $\tilde{l} = w_u g(l)$ ,  $\tilde{u} = w_u g(u)$ .  $\tilde{a}^{\leq}$  is given by the point  $(l, w_u g(l))$  and the minimum slope of all slopes of  $w_l g(x)$  and  $w_u g(x)$  on  $[l, u]$ , i.e.,  $\tilde{a}^{\leq} = w_u g(l) + \min(w_u g'(x) |_{x=u}, w_u g'(x) |_{x=l}, w_l g'(x) |_{x=u}, w_l g'(x) |_{x=l})(x-l) = w_u g(l) + \min(w_l g'(x) |_{x=u}, w_l g'(x) |_{x=l})(x-l) = w_u g(l) + w_l \kappa'(x-l)$ ; Similarly,  $\tilde{a}^{\geq}$  is given by the point  $(u, w_u g(u))$  and the same minimum slope, i.e.,  $\tilde{a}^{\geq} = w_u g(u) + w_l \kappa'(x-u)$ .
- When  $w_u \leq 0$  (cf. Figure 12(b)):
  - If  $l \geq 0$ , then  $\tilde{l} = w_l g(u)$ ,  $\tilde{u} = w_u g(l)$ ,  $\tilde{a}^{\leq} = a_{w_l}^{\leq}$ , and  $\tilde{a}^{\geq} = a_{w_u}^{\geq}$ .  $a_{w_l}^{\leq}$  is given by function defined by the point  $(u, w_l g(u))$  and the maximum slope  $\max(w_l g'(l), w_l g'(u)) = w_l \kappa'$ , i.e.,  $\tilde{a}^{\leq} = w_l g(u) + w_l \kappa'(x-u)$ .  $a_{w_u}^{\geq}$  is given by line segment defined by two points  $(l, w_u g(l))$  and  $(u, w_u g(u))$ , i.e.,  $\tilde{a}^{\geq} = w_u g(l) + w_u \kappa(x-l)$ ;
  - If  $u \leq 0$ , then  $\tilde{l} = w_u g(u)$ ,  $\tilde{u} = w_l g(l)$ ,  $\tilde{a}^{\leq} = a_{w_u}^{\leq}$ , and  $\tilde{a}^{\geq} = a_{w_l}^{\geq}$ .  $a_{w_u}^{\leq}$  is given by line segment by two points  $(l, w_u g(l))$  and  $(u, w_u g(u))$ , i.e.,  $\tilde{a}^{\leq} = w_u g(u) + w_u \kappa(x-u)$ .  $a_{w_l}^{\geq}$  is given by function defined by the point  $(l, w_l g(l))$  and the maximum slope of  $w_l g(x)$  on  $[l, u]$ , i.e.,  $\tilde{a}^{\geq} = w_l g(l) + w_l \kappa'(x-l)$ ;
  - If  $l < 0 < u$ , then  $\tilde{l} = w_l g(u)$ ,  $\tilde{u} = w_l g(l)$ .  $\tilde{a}^{\leq}$  is given by the point  $u, w_l g(u)$  and the maximum slope of all slopes of  $w_l g(x)$  and  $w_u g(x)$  on  $[l, u]$ , i.e.,  $\tilde{a}^{\leq} = w_l g(u) + \max(w_l g'(x) |_{x=u}, w_l g'(x) |_{x=l}, w_u g'(x) |_{x=u}, w_u g'(x) |_{x=l})(x-u) = w_l g(u) + \max(w_u g'(x) |_{x=u}, w_u g'(x) |_{x=l})$

$(x - u) = w_l g(u) + w_u \kappa'(x - u)$ ; Similarly,  $\tilde{a}^{\geq}$  is given by the point  $(l, w_l g(l))$  and the same maximum slope, i.e.,  $\tilde{a}^{\geq} = w_l g(l) + w_u \kappa'(x - l)$ .

Finally, given the weighted Tanh function  $\vec{W}_{j,k}^{i+1} \cdot \text{Tanh}(x_{k,1}^i)$ , we can construct and obtain its sound abstract element presented in Table 1.  $\square$

### C Illustration of Interval Partition Effectiveness

This section presents an illustrative example (see Figure 13) to explain how the interval partition, as part of the binary search strategy, can enhance abstraction precision, albeit to a limited extent. Specifically, the blue-shaded region in Figure 13(a) represents the value domain of the abstract element when abstraction is directly applied considering  $w_l \leq w \leq w_u$  with  $w_l < 0 < w_u$ . In contrast, the green and red regions, shown in Figures 13(b) and 13(c), respectively, illustrate the value domains of the abstract elements obtained by separately applying abstraction concerning  $w_l \leq w \leq 0$  and  $0 \leq w \leq w_u$ . By combining the attraction results after interval partition, we can find that the final union of value domains, as depicted in Figure 13(d), is more precise (i.e., smaller) compared to that in Figure 13(a). This demonstrates an improvement in abstraction precision achieved through the interval partition strategy. The ground truth of the abstraction in Figure 13(e).

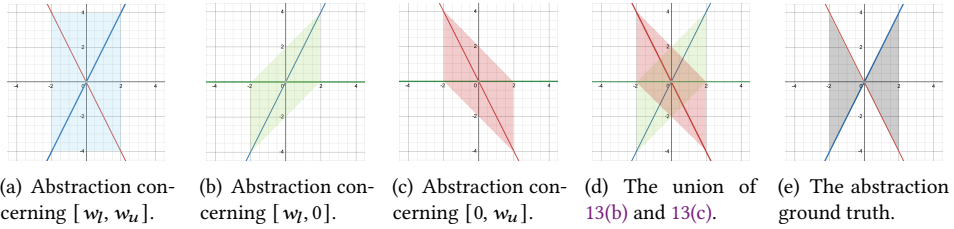


Fig. 13. An illustration example explaining why interval partitions can enhance the abstraction precision when  $w_l < 0 < w_u$  for the weighted input neuron  $\vec{W}_{j,k}^2 x_k^i$  with  $\vec{W}_{j,k}^2 \in [w_l, w_u]$ .

### D Additional Experimental Results

This section presents the details of the experimental results by BFAVerifier without Binary Search in BFA\_RA, which are omitted in Section 5.2 and Section 5.3.

Table 12. Verification results of BFAVerifier without Binary Search in BFA\_RA on ACAS Xu.

Property	BFA_RA		BFA_MILP		AvgTime(s)		#TO
	#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP	
Prop_3_WL	99.7%	0	0	24	353.4	34.6	0
Prop_3_WR	99.8%	0	1	26	355.7	8.4	0
Prop_3_SL	99.8%	0	0	24	356.5	15.3	0
Prop_3_SR	99.5%	0	0	27	357.3	176.3	0
Prop_5_SR	97.4%	0	0	18	379.0	376.3	3
Prop_10_COC	99.3%	9	1	10	361.7	84.7	22

Table 13. Verification results of BFAVerifier without Binary Search in BFA\_RA on MNIST for small networks when  $(Q, r, n) = (8, 0, 1)$ .

Network	BFA_RA		BFA_MILP		AvgTime(s)		#TO
	#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP	
3blk_10	99.7%	0	0	20	30.8	0.2	0
3blk_30	99.9%	11	0	9	103.4	0.4	0
3blk_50	99.9%	19	1	0	204.7	0.6	0
5blk_10	99.2%	0	0	20	47.1	0.4	0
5blk_30	99.9%	17	3	0	171.4	0.4	0
5blk_50	99.9%	0	0	20	349.5	0.8	0

Table 14. Detailed verification results of BFAVerifier without Binary Search in BFA\_RA on 3blk\_100 and 5blk\_100 with  $Q \in \{4, 8\}$ .

	r	n	BFA_RA		BFA_MILP		AvgTime(s)		#TO	
			#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP		
Q = 4	0	1	100.0%	40	0	0	727.1	0	0	
		2	100.0%	40	0	0	747.7	0	0	
		4	100.0%	40	0	0	760.8	0	0	
	2	1	100.0%	40	0	0	2025.1	0	0	
		2	99.9%	38	1	1	2058.7	6.5	0	
		4	99.9%	38	1	1	2034.2	6.8	0	
	4	1	99.8%	29	3	0	2037.0	1038.9	8	
		2	99.7%	19	3	3	2044.2	694.5	15	
		4	99.8%	19	2	2	2032.4	103.0	17	
		r	n	BFA_RA		BFA_MILP		AvgTime(s)		#TO
				#Safe_Paras	#Proved	#Proved	#Falsified	BFA_RA	BFA_MILP	
	Q = 8	0	1	100.0%	40	0	0	717.6	0	0
2			100.0%	40	0	0	738.6	0	0	
4			99.9%	39	0	1	755.5	1.7	0	
2		1	99.9%	38	1	1	2036.7	2.7	0	
		2	99.9%	37	1	2	2029.5	8.1	0	
		4	99.9%	36	0	3	2027.9	4.6	1	
4		1	99.5%	22	3	1	2046.6	1419.0	14	
		2	99.3%	18	3	1	2036.1	858.4	18	
		4	99.4%	18	1	3	2067.3	295.6	18	